MICRO-ORDINATEURS

LA CONDUITE DU TO 7

Jean François TERRAL

Tome 1

LA CONDUITE DU TO7

1. Basic
Crayon optique
Langage machine

DANS LA MÊME COLLECTION

SCHOMBERG - Le Basic Universel.
SCHOMBERG - Micro-ordinateurs : Comment ça marche ?
HERNANDEZ - Pascal par l'exemple.
NOLLET - Le conduite du ZX 81.
PELLIER - La conduite du TRS 80.
LADEVIE - Votre gestion avec BASIC sur micro-ordinateur.
QUEINNEC - Langage d'un autre type : LISP.
PELLIER - Programmez vos jeux d'action rapide sur TRS 80.
ASTIER - La conduite du l'APPLE II
Tome 1 : le Basic de l'APPLE II
Tome 2 : le système graphique et l'assembleur de l'APPLE II.
MONTEIL - L'assembleur facile du 6502.
LEPAPE - L'assembleur facile du Z 80.
OROS et - ZX 81 à la conquête des jeux.
PERBOST - CASSETTE - ZX 81 à la conquête des jeux.
PERBOST - CASSETTE N° 2 - 13 jeux 1 K.
DAX - CP/M et sa famille.
NOLLET - Langage machine, trucs et astuces sur ZX 81.
BICKING - La conduite du PC 1211 (ou TRS 80 pocket).
TEJA - Apprenez à parler à votre ordinateur.
MONTEIL - La conduite du VIC 20.
PLOUIN - La conduite de l'IBM-PC.
SAGUEZ - Télécommande avec votre micro-ordinateur.
PELLIER - Tout sur les disques du TRS 80 modèles I et III.
OROS et PERBOST - La conduite du FX-702 P et FX-801 P.
GROS - La conduite du PC 1500
HARWOOD - Jeux et applications pour ZX SPECTRUM
HARTNELL - Le grand livre du ZX SPECTRUM
HARTNELL et JONES - La conduite du ZX SPECTRUM
VULDY - Graphisme 3 D sur votre micro-ordinateur
BOUQUEROD - Des extensions à construire pour votre ZX 81
PINSON - Le Basic en gestion sur Apple II
WILLARD - La conduite du TI 99
CEYRAT - Mon TI 99/4A
DELANNOY - Les fichiers en Basic ... sur micro-ordinateur
AUBERT - Pratiquez l'intelligence artificielle
PERBOST et MASSE - VIC 20 à la conquête des jeux
TERRAL - La conduite du TO7.
ASTIER - La conduite de l'ORIC-1.
MONTEIL - Apprenez à programmer en LOGO.
MONTEIL - La conduite du COMMODORE 64.
Tome 1 : Basic, graphisme et son.

LOGILIVRES EYROLLES
(logiciels sur cassettes)

PELLIER - Kemikaze (jeu pour ZX Spectrum).
PELLIER - Astéroïdes (jeu pour ZX Spectrum).
PELLIER - Othello/Isola (jeux pour ZX Spectrum.
PELLIER - Éditeur/Assembleur pour ZX Spectrum.
PERBOST et MASSE - VIC 20 à la conquête des jeux.
(VIC 20 version de base).

LA CONDUITE DU TO7

1. Basic
Crayon optique
Langage machine

Jean-François TERRAL

Collection animée
par Richard SCHOMBERG

EYROLLES
61, boulevard Saint-Germain — 75005 Paris
1984

Si vous désirez être tenu au courant de nos publications, il vous suffit d'adresser votre carte de visite au :

Service «Presse», Éditions EYROLLES
61, Boulevard Saint-Germain,
75240 PARIS CEDEX 05,

en précisant les domaines qui vous intéressent. Vous recevrez régulièrement un avis de parution des nouveautés en vente chez votre libraire habituel.

«La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que les «copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective» et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, «toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite» (alinéa 1er de l'article 40)».

«Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles 425 et suivants du Code pénal».

© Éditions EYROLLES, 1984

INTRODUCTION

LA CONDUITE DU TO7 s'adresse à tous ceux qui possèdent déjà une connaissance du langage BASIC (ou de tout autre langage) et qui utilisent ou envisagent d'utiliser un TO7.

Ceux qui, abordant pour la première fois l'informatique, désirent apprendre le langage BASIC ou se familiariser avec le fonctionnement interne des ordinateurs, liront avec profit le BASIC UNIVERSEL et MICRO-ORDINATEURS : COMMENT ÇA MARCHE ? publiés dans la même collection. La lecture de ces deux livres de base vous permettra d'aborder sans aucune difficulté le manuel BASIC de n'importe quel micro-ordinateur.

Dans ce premier tome nous abordons le BASIC "en général". Les multiples possibilités graphiques de la machine, la façon de les exploiter au maximum et les extensions seront exposées dans un deuxième tome.

— *Dans le premier chapitre du présent ouvrage vous reprenons toutes les commandes, instructions et fonctions du BASIC TO7, chacunes d'entre elles étant illustrées par un ou plusieurs exemples.*

— *Le chapitre 2 traite de l'utilisation du crayon optique.*

— *Au chapitre 3 nous expliquons comment créer vos propres programmes en langage machine avec échanges de paramètres entre le programme BASIC et le programme écrit en langage machine.*

Table des matières

Introduction VII

1. Les instructions BASIC du TO7 1

1.1. Généralités 1
1.2. Les Commandes du BASIC TO7 8
1.3. Les Instructions et fonctions du BASIC TO7 18

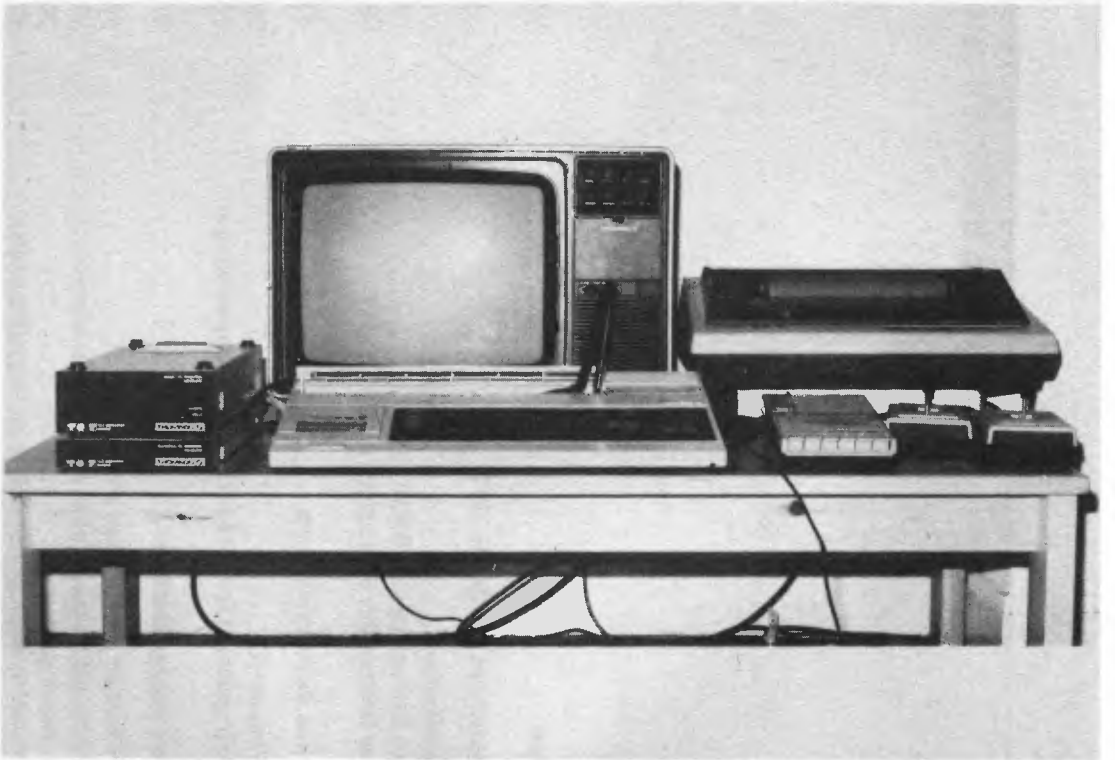
2. Utilisation du crayon optique 77

2.1. Généralités 78
2.2. Instructions et fonctions relatives au crayon optique 81

3. Accès au langage machine 91

3.1. Utilisation d'un langage assembleur 91
3.2. Création d'un programme en langage machine 92
3.3. Quelques précautions 96
3.4. Exécution du programme écrit au langage machine 97
3.5. DEFUSR et USR 100

Annexe — Code des instructions du 6809 108



TO7 muni de toutes ses extensions.

1

Les instructions BASIC du TO7

Ce chapitre s'adresse surtout à ceux qui connaissent déjà le langage BASIC. Après avoir abordé quelques généralités sur le BASIC TO7, nous reprendrons en les accompagnant d'un exemple, les commandes, les instructions et les fonctions du TO7. Celles-ci ont été classées par ordre alphabétique afin de pouvoir être retrouvées rapidement.

1.1. GÉNÉRALITÉS

1.1.1. Structure des instructions

Le Basic TO7 accepte des numéros d'instruction compris entre Ø et 63 999.

Une ligne de programme ou ligne logique peut contenir 255 caractères maximum.

Une ligne d'affichage sur l'écran ou ligne physique a une longueur de 4Ø caractères.

Plusieurs instructions par ligne sont autorisées. Celles-ci devront être séparées par le caractère " : ".

Ainsi une ligne programme occupera au maximum six lignes et demi sur l'écran.

Exemple: La "ligne programme" suivante comporte quatre instructions et a une longueur de 69 caractères.

```
1Ø FOR I = Ø TO 1Ø : PRINT " LE CARRE DE ";X;
   " EST "; X * X : NEXT I : END
```

Il n'est pas obligatoire de séparer le mot clé et les paramètres d'une instruction par un espace. Ainsi l'instruction:

```
1Ø IF A = Ø THEN PRINT "BON" ELSE C = 1
```

pourra aussi être écrite de la façon suivante:

```
1Ø IFA = ØTHENPRINT"BON"ELSE C = 1
```

qui est, malgré son manque total de clarté, syntaxiquement correcte. Cependant il faudra obligatoirement insérer un espace lorsqu'un nom de variable sera suivi par un mot clé.

Exemple: le programme suivant affiche le reste de la division par 4, de la variable I.

```
1Ø CLS
2Ø LET I = 7
3Ø PRINT I MOD 4
```

Lancer l'exécution du programme par RUN. Le résultat affiché est 3. Supprimer maintenant l'espace entre I et MOD. La ligne 3Ø devient donc.

```
3Ø PRINT IMOD 4
```

Relancer l'exécution du programme. Le résultat affiché est 0.4.

En effet, en ayant supprimé l'espace entre I et MOD, le TO7 ne détecte plus qu'il doit afficher le reste de la division de la variable I par 4. Pour lui, IMOD est un nom de variable. Celle-ci n'ayant jamais été initialisée, sa valeur est supposée être nulle.

— Les instructions PRINT et REM peuvent être remplacées respectivement par les caractères ? et ' (apostrophe).

— L'exécution d'un programme peut être arrêtée en appuyant simultanément sur les touches CNT et C (ceci correspond à la touche BREAK sur d'autres machines).

— De même les touches CNT et X effacent le reste de la ligne courante.

1.1.2. Variables

Le BASIC du TO7 accepte quatre types de variables: entières, réelles simple précision, réelles double précision et alphanumériques. Les noms de variables peuvent avoir une longueur quelconque (en réalité la longueur maximale sera de 253 caractères) toutefois le BASIC ne tiendra compte que des 16 premiers caractères.

— *variables entières*: Elles sont identifiées par le caractère %, leur valeur doit être comprise entre — 32 768 et + 32 767.

Exemple:

```
10 AN% = 1 982
20 B% = 32 767
30 C% = — 32 768
```

— *variables réelles simples précision.* C'est le type de variable pris par défaut. On peut cependant les identifier par le caractère "!" Leur valeur doit être comprise entre 10^{-38} et 10^{+38}. Leur précision est de six chiffres significatifs.

Exemple:

```
10 TOTAL = 1 240.27
20 TVA I = 18.6
```

— *variables réelles double précision.* Elles sont identifiées par le caractère # leur valeur doit être comprise entre 10^{-38} et 10^{+38}. Leur précision est de quinze chiffres significatifs.

Exemple:

```
1Ø A# = Ø.123456789
2Ø DELTAX# = 3.5642 D — 3
```

— *variables alphanumériques.* Identifiées par le caractère **$**, leur longueur ne peut excéder 255 caractères.

Exemple:

```
1Ø A$ = "JE SUIS LE TO7 DE THOMSON"
2Ø MOIS$ = "DECEMBRE"
```

Remarque: On peut reprendre un nom identique pour des variables ayant un type différent. LE BASIC TO7, les identifiera comme des variables différentes.

Exemple:

```
1Ø A% = 1982 : A = 125.67
2Ø PRINT A%, A
```

Remarque : Une valeur non entière, affectée à une variable entière, sera arrondie à la valeur entière la plus proche.

Ainsi:

AN% = 1982.3 affecte à la variable entière AN la valeur 1982
AN% = 1982.5 affecte à la variable entière AN la valeur 1983
AN% = 1982.7 affecte à la variable entière AN la valeur 1983

1.1.3. Constantes

Comme les variables elles peuvent être de type entier, réelle, simple précision, réelle double précision et alphanumérique.

Exemple:

```
A% = 1 235%
TOTAL = 12.5Ø%
VARIATION# = Ø.ØØØØ 12345#
```

Remarque :

1. La notation sous forme de puissance de 1Ø peut être utilisée. E pour les constantes réelles simple précision et D pour les constantes réelles double précision. Ainsi en simple précision 1ØØ peut s'écrire 1EØ2 et Ø.ØØ1 peut s'écrire 1E — Ø3.

   On peut aussi remarquer la différence de résultat entre :

   ```
   A   = Ø.12345 E — Ø3 : PRINT A          (simple précision)
   et A# = Ø.12345 D — Ø3 :PRINT A#        (double précision)
   ```

2. Les constantes peuvent être exprimées en binaire, octal, décimal et hexadécimal. Les constantes binaires, octales et hexadécimales seront respectivement identifiées par les caractères &B, &O, &H placés devant la valeur.

Exemple : Dans le programme suivant, la constante décimale 16 est exprimée sous sa forme binaire, octale et hexadécimale.

```
1Ø C = &B 1ØØØØ
2Ø D = &O 2Ø
3Ø E = &H 1Ø
4Ø PRINT C, D, E
```

1.1.4. Tableaux

Les tableaux peuvent avoir jusqu'à 255 dimensions, chaque dimension comportant au maximum 32 767 éléments.

Exemple :

```
A (1, 2, 3, 4)
GRILLE (2Ø, 24)
```

1.1.5. Opérateurs

Le BASIC TO7 possède trois types d'opérateurs.

1.1.5.1. Les opérateurs arithmétiques

Ce sont : l'élévation à la puissance (touche SHIFT +'â/) la multiplication (*), la division (/), l'addition (+), la soustraction (–), la division entière (@) et le modulo (MOD).

Remarque : La division entière donne le quotient de la division et le modulo le reste.

Ainsi : 75 @ 1Ø donne 7 quotient de la division de 75 par 1Ø
et 75 MOD 1Ø donne 5 reste de la division de 75 par 1Ø

1.1.5.2. Les opérateus da relation

Ce sont :

> *supérieur à,* >= supérieur ou égal à
= égal à, =< inférieur ou égal à,
< inférieur à et <> différent de.

1.1.5.3. Les opérateurs logiques

Ces opérateurs travaillent sur 16 bits. Ce sont : le ET logique (AND), le OU logique (OR), le complément (NOT), le OU exclusif (XOR), l'implication (IMP) et l'équivalence (EQV).

Pour les fonctions ET, OU, OU exclusif et complément, reportez vous au livre "LE BASIC UNIVERSEL" paru dans la même collection Détaillons le fonctionnement des fonctions implication (IMP) et équivalence (EQV).

si A = Ø et B = Ø alors A IMP B donne 1 comme résultat
si A = Ø et B = Ø alors A EQV B donne 1 comme résultat
si A = Ø et B = 1 alors A IMP B donne 1 comme résultat
si A = Ø et B = 1 alors A EQV B donne Ø comme résultat
si A = 1 et B = Ø alors A IMP B donne Ø comme résultat
si A = 1 et B = Ø alors A EQV B donne Ø comme résultat
si A = 1 et B = 1 alors A IMP B donne 1 comme résultat
si A = 1 et B = 1 alors A EQV B donne 1 comme résultat

Exemple: Cherchons quel serait le résultat donné par le TO7 si nous lui demandions d'évaluer les deux expressions:

156 IMP 35 et 156 EQV 35

Tout d'abord écrivons les nombres 156 et 35 en binaire sur 16 bits:

	156 =	ØØØØ ØØØØ 1ØØ1 11ØØ		
et	35 =	ØØØØ ØØØØ ØØ1Ø ØØ11		
	numéro du bit	15	7 5	Ø

Prenons le cas de l'implication. Appliquons bit à bit les "règles" de calcul énoncées ci-dessus:

Pour le bit 15 A = Ø et B = Ø donc le résultat est 1

Pour le bit 7 A = 1 et B = Ø donc le résultat est Ø

Pour le bit 5 A = Ø et B = 1 donc le résultat est 1

en effectuant l'évaluation sur les 16 bits nous obtiendrons le résultat suivant:

1111 1111 Ø11Ø ØØ11

c'est-à-dire le nombre — 157.

En procédant de manière identique pour l'opérateur logique EQV, le résultat sera:

1111 1111 Ø1ØØ ØØØØ

c'est-à-dire le nombre — 192.

Nous pouvons vérifier ces deux résultats en entrant, en mode commande, les deux ordres:

PRINT 156 IMP 35
et PRINT 156 EQV 35

1.1.5.4. Priorité des opérateurs

La priorité des opérateurs est la suivante :

↑	élévation à la puissance
–	moins unaire
*, /	multiplication division
@	division entière (quotient)
MOD	modulo (reste)
+,–	addition soustraction

Opérateurs de relations (tous les opérateurs de relation ont la même priorité).

Opérateurs logiques (tous les opérateurs logiques ont la même priorité).

Les expressions arithmétiques et logiques sont évaluées de gauche à droite selon les priorités des opérateurs. L'ordre des priorités peut être changé par l'emploi de parenthèses.

Exemple : L'expression :

3 + 4 * 2 MOD 3 + 10 sera évaluée comme suit
3 + 8 MOD 3 + 10
3 + 2 + 10 = 15

tandis que : 3 + 4 * (2 MOD 3 + 10) sera évaluée :
3 + 4 * (2 + 10)
3 + 4 * 12 = 51

1.2. LES COMMANDES DU BASIC TO7

Dans ce paragraphe nous reprenons les commandes du BASIC du TO7. Toutes ces commandes peuvent être utilisées comme des instructions vous les retrouverez donc dans le paragraphe Instructions.

AUTO: Commande

AUTO n, i

n est un numéro de ligne
i est un incrément.

La commande AUTO provoque une numérotation automatique des lignes de votre programme BASIC. La première ligne aura le numéro n, la seconde le numéro n + i, la troisième le numéro n + 2i et ainsi de suite. Chaque ligne est validée en appuyant sur la touche ENTREE.

Si n et i sont omis, ils seront alors forcés à 1Ø.

Si le numéro de ligne existe déjà il sera suivi d'un astérisque. Vous pouvez alors soit appuyer sur la touche ENTREE si vous ne désirez pas modifier cette ligne, soit taper une nouvelle instruction ; l'instruction précédente sera alors perdue.

Pour arrêter la numérotation automatique, appuyer sur les touches CNT et C simultanément.

Exemples :

— AUTO 6, 8 numérotation des lignes de 8 en 8, le premier numéro utilisé étant 6.

— AUTO numérotation de 10 en 10, le premier numéro utilisé étant 10.

Remarque 1: La forme AUTO n, sans incrément provoquera une numérotation automatique commençant à la ligne n en reprenant l'incrément spécifié précédemment.

Exemple: Entrez la commande AUTO 3, 15. La numérotation se fait de 15 en 15 puis faites CNT C pour arrêter la numérotation. Tapez alors AUTO 6Ø, : la numérotation reprend à partir du numéro de ligne 60 et se fait de 15 en 15. Si vous aviez entré AUTO, la numérotation aurait repris à partir de la ligne 10 avec un incrément de 15.

Remarque 2: Il n'est pas possible de reprendre implicitement la numérotation à partir du dernier numéro de ligne utilisé. Il faudra toujours spécifier clairement ce numéro dans la commande sinon la valeur 10 sera prise par défaut.

CONT: Commande.

La commande CONT permet de reprendre l'exécution d'un programme à partir de l'instruction à laquelle il s'est arrêtée. (Arrêt dû à une instruction STOP ou END ou à un appui sur les touches CNT C).

Il est possible avant de reprendre l'exécution d'afficher ou de modifier les valeurs des variables du programme. Cependant la **modification d'une ligne quelconque du programme est interdite.**

Exemple:

```
1Ø FOR I = 1 TO 5
2Ø STOP
3Ø NEXT I
```

Chaque fois que le programme s'arrête, nous pouvons demander la valeur de la variable I, en entrant la commande PRINT I. Nous pourrions aussi modifier la valeur de cette variable en faisant I = 1Ø par exemple.

Par contre si nous avions voulu remplacer l'instruction 2Ø STOP par 2Ø PRINT I, il n'aurait pas été possible de reprendre l'exécution par la commande CONT.

Remarque: La commande CONT ne permet pas de reprendre l'exécution du programme lorsque l'arrêt de celle-ci a été provoqué par une erreur.

DELETE: Commande.

DELETE n-m

n et m sont des numéros de lignes.

La commande DELETE supprime toutes les lignes de programmes dont le numéro est compris entre n et m. Les lignes de numéro n et m seront aussi supprimées. n et m peuvent être omis. Si n est omis, le numéro pris par défaut est celui de la première ligne du programme. Si m est omis, le numéro pris par défaut est celui de la dernière ligne du programme. Si n est spécifié seul et sans tiret alors seule la ligne de numéro n sera supprimée.

Exemple:

DELETE 2Ø – 1ØØ	supprime toutes les lignes de 2Ø à 1ØØ
DELETE – 1ØØ	supprime toutes les lignes du début du programme à la ligne 1ØØ
DELETE 1ØØ –	supprime toutes les lignes de la ligne 1ØØ à la fin du programme
DELETE 1ØØ	supprime la ligne 1ØØ uniquement

Remarque:

DELETE — supprime toutes les instructions du programme.

GOTO: Commande.

GOTO n

n est un numéro de ligne.

La commande GOTO permet de reprendre l'exécution d'un programme à partir de la première instruction de la ligne de programme dont le numéro est n. Toutefois, contrairement à la commance RUN n, les variables ne seront pas remises à zéro.

(Voir aussi la commande RUN).

LIST: Commande.

LIST: n-m

n et m sont des numéros de ligne.

La commande LIST permet d'obtenir le listage d'une partie ou de tout le programme contenu en mémoire. n et m ont la même signification que dans la commande DELETE. Reportez-vous donc à cette commande.

Exemple:

LIST 1Ø – 2ØØ	liste le programme de la ligne 1Ø à la ligne 2ØØ
LIST – 2ØØ	liste le programme du début à la ligne 2ØØ
LIST 5Ø –	liste le programme de la ligne 5Ø jusqu'à la fin
LIST 5Ø	liste la ligne 5Ø uniquement

Remarque :

1) Le listage peut être arrêté momentanément en appuyant sur la touche STOP. Un appui touche quelconque le fera continuer.

2) Le listage peut être arrêté définitivement en appuyant sur les touches CNT C

LOAD: Commande.

LOAD "nom", R

nom est le nom donné au programme lors de la sauvegarde.

La commande LOAD permet de recharger le programme dont le nom est spécifié. La mise en route et l'arrêt du magnétophone sont effectués automatiquement par le TO7. Sur l'écran seront affichés les noms des différents fichiers (programme ou données) trouvés avant le programme demandé. Quand le programme est chargé, la mention OK apparaît sur l'écran.

Si le nom est omis, le programme chargé sera le premier programme rencontré sur la cassette.

Si au moment du chargement des fichiers sont ouverts, ils seront refermés par la commande LOAD.

Il est possible de lancer l'exécution du programme immédiatement après le chargement. Pour cela, il suffit de spécifier l'option ,R après le nom du programme. Lorsque l'option R est présente, les fichiers ouverts ne sont pas refermés avant le chargement.

Exemple 1: Chargement du programme dont le nom est PROG 1. Les fichiers ouverts sont refermés.

```
LOAD "PROG 1"
```

Exemple 2: Chargement et exécution du programme dont le nom est CALCUL. Les fichiers ouverts ne sont pas refermés.

```
LOAD "CALCUL" ,R
```

Remarques:

1) Il n'est pas possible d'arrêter le chargement en appuyant sur les touches CNT C. Il faudra alors systématiquement utiliser le bouton

RESET. Ceci peut être utilisé pour obtenir le nom de tous les programmes stockés sur une cassette. En effet il suffit de demander le chargement d'un programme qui n'existe pas.

2) Le nom du programme doit être le même que celui donné lors de la sauvegarde. Reportez-vous à la commande SAVE pour les règles concernant le nom du programme. Si les trois caractères de suffixe ne sont pas précisés, ils seront forcés à BAS. Ainsi les deux commandes.

```
LOAD "CALCUL"
```
et `LOAD "CALCUL.BAS"`

sont identiques.

LOADM: Commande.

LOADM "nom" ,d ,R

nom est le nom d'un programme écrit en langage machine
d est un déplacement.

La commande LOADM permet de recharger un programme en langage machine qui avait été sauvegardé sur cassette. Si le déplacement d est omis, le programme sera rechargé à l'adresse à laquelle il se trouvait lorsqu'il a été sauvegardé. Si d est spécifié, le programme sera rechargé à l'adresse obtenue en ajoutant le déplacement d à l'adresse à laquelle le programme se trouvait lors de la sauvegarde.

Comme pour la commande LOAD, l'option R peut être utilisée. L'exécution du programme en langage machine débutera alors dès que le chargement sera terminé.

Exemple: Le programme suivant charge un programme en langage machine à l'adresse 32 700

```
10 I=32699
20 CLEAR ,I
25 I=32 699
30 FOR X=I+1 TO I+5
40 READ A: POKE I, A
50 NEXT X
60 DATA &H4F, &HB7, &H7F, &HC6, &H39
```

CLRA, STA 127 198 RTS

On peut vérifier que le programme en langage machine est bien à l'adresse 32 7ØØ avec le programme BASIC suivant.

```
1ØØ I=32 7ØØ
11Ø FOR X=I TO I+4
12Ø PRINT HEX$ (PEEK(I)) ; " " ;
13Ø NEXT X
```

Sauvegardons le programme en langage machine. Pour cela nous entrons la commande.

```
SAVEM "MC1", 32 7ØØ, 32 7Ø4, 32 7ØØ
```

Après avoir fait NEW rechargeons le programme en langage machine à l'adresse 32 72Ø c'est-à-dire 2Ø octets plus loin par rapport à son adresse initiale.

```
CLEAR ,32 719
LOADM "MC1" ,2Ø
```

Nous pouvons vérifier que le programme en langage machine a bien été rechargé à l'adresse 32 72Ø. Pour cela il suffit de réutiliser le programme BASIC précédent en remplaçant la ligne 1ØØ par:

```
1ØØ I = 32 72Ø
```

Remarques: Si lors de la sauvegarde aucune extension n'est précisée dans "nom", le BASIC TO7 prendra par défaut l'extension. BIN.

Voir aussi la commande SAVEM.

MERGE: Commande.

MERGE "nom" ,R

nom est le nom d'un programme qui a été sauvegardé en ASCII

La commande MERGE fusionne le programme situé en mémoire avec le programme sauvegardé sur cassette ou sur disquette et dont le nom est spécifié.

La fusion est réalisée suivant les numéros de ligne. Ainsi si deux numéros de ligne sont identiques, la ligne conservée sera celle du programme sauvegardé sur la cassette ou la disquette.

L'option R permet de lancer l'exécution du programme obtenu dès que la réunion des deux programmes a été réalisée.

Exemple: Sauvegardons le programme suivant sous le nom PROG 2.

```
15 PRINT "PROG 2"
2Ø REM PROGRAMME DEVANT ETRE SAUVEGARDE EN ASCII
3Ø PRINT "LIGNE 3Ø - PROG 2"

SAVE "PROG 2" ,A
```

Entrons le programme ci-dessous, après avoir tapé la commande NEW.

```
 5 REM PROG 1
1Ø PRINT "PROG 1"
2Ø STOP
```

Pour fusionner les deux programmes il suffira de rentrer la commande.

```
MERGE "PROG 2"
```

ou `MERGE "PROG 2" ,R` pour faire exécuter le nouveau programme immédiatement.

Remarquer que la ligne 2Ø STOP a été remplacée par la ligne 2Ø REM PROGRAMME DEVANT ETRE SAUVEGARDE EN ASCII.

NEW: Commande.

```
NEW
```

La commande NEW efface le programme et détruit toutes les variables. Cette commande doit être utilisée avant l'entrée d'un nouveau programme BASIC.

MOTORON, MOTOROFF: Commandes.

```
MOTORON
MOTOROFF
```

Ces deux commandes permettent respectivement la mise en marche et l'arrêt du magnétophone.

RUN: Commande.

RUN n

n est un numéro de ligne.

Cette commande permet de lancer l'exécution d'un programme à partir de la première instruction de la ligne de programme dont le numéro est n. Cette commande referme tous les fichiers ouverts et remet à zéro toutes les variables du programme. Si l'on désire conserver le contenu des variables, il faut utiliser la commande GOTO.

Si le numéro de ligne est omis, l'exécution commencera au début du programme.

Il est possible de faire exécuter un programme sauvegardé sur cassette (ou disquette) avec la commande RUN "nom". Le programme dont le nom est donné sera recherché, chargé en mémoire, puis exécuté.

SAVE: Commande

SAVE "nom", A ou P

nom est le nom du programme.

La commande SAVE permet de sauvegarder un programme sur une cassette ou une disquette.

Le nom du programme ne pourra pas dépasser 11 caractères: 8 caractères plus trois caractères de suffixe séparés des autres caractères par un point. Si les trois caractères de suffixe ne sont pas précisés, ils seront forcés à "BAS". Ainsi la commande SAVE "PROG" donnera en réalité le nom PROG.BAS à votre programme.

L'option A permet de sauvegarder le programme en ASCII. Utilisez la **obligatoirement** si vous désirez fusionner ce programme avec un autre par la suite (commande MERGE). (ou LIST"CASS:----"

L'option P permet de "protéger" votre programme. Ainsi, après l'avoir rechargé, il sera impossible de le lister, d'insérer des nouvelles lignes ou d'utiliser les commandes AUTO, PEEK, POKE et SAVE.

Exemple:

```
SAVE "PROG 1"
SAVE "CYCLO.IDE", A
SAVE "ESS", P
```

Remarques:

1) Il est possible de "resauvegarder" un programme protégé, mais il faut pour cela obligatoirement le sauvegarder en mode protégé en utilisant l'option P, ce qui ne nous avance pas énormément!...

2) Un programme protégé peut toutefois contenir des **instructions** PEEK et POKE. Seules les **commandes** PEEK et POKE sont interdites.

3) Si les trois caractères extension du nom du programme sont présents, ils devront être indiqués dans la commande LOAD. Si ces trois caractères ont été omis ou forcés à .BAS il ne sera pas nécessaire de les rappeler dans la commande LOAD. Ainsi un programme sauvegardé par la commande SAVE "PROG 1.ESS" devra être rechargé par la commande LOAD "PROG 1.ESS" tandis qu'un programme sauvegardé par la commande SAVE "PROG1.BAS" ou SAVE "PROG 1" pourra être rechargé par LOAD "PROG 1".

SAVEM: Commande

SAVEM "nom", adresse début, adresse fin, adresse exécution.

nom est un nom de programme écrit en langage machine.
adresse début est l'adresse de début du programme écrit en langage machine
adresse fin est l'adresse de fin du programme écrit en langage machine
adresse exécution est l'adresse à laquelle commence l'exécution du programme écrit en langage machine (point d'entrée).

La commande SAVEM permet de sauvegarder un programme écrit en langage machine sur cassette ou sur disquette. Le nom du programme suivra les règles indiquées dans la commande SAVE. Toutefois si les caractères de suffixe ne sont pas précisés, ils seront forcés à .BIN.

Exemple:

```
SAVEM "MC1", 32 7ØØ, 32 715, 32 7Ø4
```

Sauvegarde du programme en langage machine situé aux adresses 32700 à 32715 et dont l'exécution commence à l'adresse 32704.

Remarque: L'adresse d'exécution est obligatoire si l'on désire utiliser la commande LOADM avec l'option R, sinon elle peut être forcée à zéro.

```
SAVEM "MC2.BIN", 32 7ØØ, 32 715, Ø
```

1.3. LES INSTRUCTIONS ET FONCTIONS DU TO7

ABS: Fonction

ABS (x)

x est un nombre exprimé en décimal, hexadécimal ou octal.
La fonction ABS donne la valeur absolue du nombre x.

Exemples:

1) Le programme suivant affiche la valeur absolue de n'importe quel nombre. Pour arrêter son exécution, donnez à x la valeur 9999.

```
10 CLS
20 INPUT"X";X
30 IF X=9999 THEN END
40 PRINT "LA VALEUR ABSLUE DE ";X;"EST:"
;ABS(X)
50 GOTO 20
```

2) Le programme suivant vérifie qu'un nombre est bien compris entre − A et + A.

```
10 CLS
20 INPUT "VALEUR LIMITE ";A
30 INPUT "NOMBRE ";X
40 IF ABS(X)>A THEN PRINT "ENTRER UN NOM
BRE COMPRIS ENTRE ";-A;"ET ";A:GOTO 30
   ELSE PRINT "CORRECT"
```

Remarque: Si x est exprimé en hexadécimal ou en octal, sa valeur sera considérée comme une valeur **non signée**. Ainsi PRINT ABS (&HFF) donnera 255 et non 1.

ASC : Fonction

ASC (c)

c est une variable alphanumérique.

La fonction ASC donne le code ASCII du premier caractère de la variable alphanumérique c. La variable alphanumérique c ne peut pas être vide.

Exemple :

Le programme suivant donne le code ASCII de n'importe quelle lettre du clavier (majuscule ou minuscule). Pour arrêter le programme répondre FIN à la question LETTRE ?

```
10 CLS
20 PRINT "ENTRER LA LETTRE DESIREE"
30 INPUT "LETTRE";L$
40 IF L$="" THEN PRINT "LA CHAINE NE PEU
T PAS ETRE VIDE." : GOTO 30
50 IF L$="FIN" THEN END
60 PRINT "LE CODE ASCII DE ";LEFT$(L$,1)
;" EST ";ASC(L$)
70 GOTO 20
```

Remarque : La fonction ASC "reconnaît" aussi les caractères graphiques utilisateur.

Ainsi le caractère graphique utilisateur Ø aura le code 128
le caractère graphique utilisateur 1 aura le code 129
le caractère graphique utilisateur 2 aura le code 130
etc
jusqu'au caractère graphique utilisateur 127 qui a le code 255

Exemple :

```
10 CLS
20 CLEAR,,3
30 DEFGR$(0)=255,0,255,0,255,0,255,0
40 DEFGR$(1)=51,51,51,51,51,51,51,51
50 DEFGR$(2)=170,85,170,85,170,85,170,85
60 FOR I=0 TO 2
70 PRINT "LE CARACTERE GRAPHIQUE UTILISA
TEUR";I;"A LE CODE";ASC(GR$(I))
80 NEXT
```

ATTRB: voir chapitre POSSIBILITÉS GRAPHIQUES (tome 2)

BEEP: Instruction

BEEP

L'instruction BEEP génère un petit "bip" sonore identique à celui qui est émis lors d'un appui touche.

Exemple:

```
10 CLS
20 INPUT "NOMBRE DE 'BIP'";N
30 FOR I=1 TO N
40   BEEP
50 NEXT I
```

BOX et **BOXF**: Voir chapitre POSSIBILITÉS GRAPHIQUES (tome 2)

CDBL: Fonction

CDBL (n)

n est une variable numérique.

La fonction CDBL convertit une variable de type entier ou simple précision en une variable du type double précision.

Exemple:

```
10 CLS
20 A% = 5 : B = 3.14 159
30 C# = CDBL (A%)
40 D# = CDBL (B)
50 PRINT A%, C#
60 PRINT B, D#
```

CHR$: Fonction

CHR$ (x)

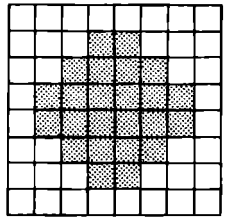
x est un nombre compris entre 0 et 255.

La fonction CHR$ renvoie le caractère dont le code est x.

— si x est compris entre Ø et 127 le caractère obtenu sera celui dont le code ASCII est x.

— si x est compris entre 128 et 255 le caractère obtenu sera le caractère utilisateur dont le numéro est $x - 128$.

Exemple : Dans le programme suivant nous créons le caractère utilisateur numéro 3 qui aura la forme suivante.



```
1Ø CLS
2Ø CLEAR,, 4
3Ø DEFGR$ (3) = Ø, & H18, & H3C, & H7E, & H7E, & H3C, & H18, Ø
4Ø PRINT CHR$ (&H48) ' lettre H
5Ø PRINT CHR$ (131) ' caractère graphique numéro 3
```

CINT : Instruction

CINT (x)

x est une variable numérique.

La fonction CINT convertit une variable de type réel simple ou double précision en une variable de type entier. Le nombre réel est arrondi au nombre entier le plus proche.

Exemple :

```
1Ø CLS
2Ø A# = 3.14159262
3Ø B = 5.45
4Ø PRINT CINT (A#), CINT (B)
```

CLEAR: Instruction

CLEAR nc, ad, ng

nc nombre d'octets à réserver pour le traitement des chaînes de caractères
ad adresse maximale utilisable par le BASIC
ng nombre de caractères graphiques

L'instruction CLEAR *efface toutes* les variables du programme. De plus elle permet:

— de modifier la place réservée par défaut pour le traitement des chaînes de caractères (300 octets).

— de réserver de la place pour venir stocker un programme écrit en langage machine en indiquant au BASIC l'adresse maximale à utiliser.

— de réserver de la place pour les caractères graphiques utilisateur.

Exemple: L'instruction suivante

```
1Ø CLEAR 2ØØ, 32 7ØØ, 4
```

diminue la place réservée pour le traitement des chaînes de caractères, (2ØØ octets au lieu des 3ØØ octets pris par défaut), réserve 67 octets pour un programme en langage machine (si vous ne disposez pas de l'extension 16 K la plus grande adresse mémoire existante est 32 767) ainsi que la place nécessaire pour déclarer quatre caractères graphiques utilisateur.

Remarques:

1) L'instruction CLEAR devra toujours être placée en tête du programme BASIC. En effet elle efface toutes les variables du programme.

2) L'adresse maximale utilisable par le BASIC est **effectivement** utilisée par celui-ci. Ainsi dans notre exemple, le sous-programme écrit en langage machine ne pourra commencer qu'à l'adresse 32 7Ø1 et non 32 7ØØ.

3) L'instruction NEW ne "récupère" pas la place réservée avec une instruction CLEAR. Ceci est aussi vrai pour le RESET.

CLOSE: Instruction

CLOSE $\#n_1$, $\#n_2$

n_1, n_2 nombre décimal compris entre 1 et 15.

L'instruction CLOSE referme les fichiers dont les numéros de canal sont n_1, n_2, etc... Si aucun numéro de canal n'est spécifié, tous les fichiers ouverts sont refermés. Le signe # est facultatif.

Exemples :

1)
```
10 CLOSE #1,3
```

2)
```
10 ON ERROR GOTO 1000
20 CLS
30 OPEN "O",#3,"DONNEES"
      |
      |
1000 PRINT "FERMETURE DU FICHIER DONNEES"
1010 CLOSE 3
1020 END
```

CLS: Instruction

CLS

L'instruction CLS efface l'écran.

Remarque: Placez toujours cette instruction en tête de vos programmes pour obtenir un écran propre surtout si vous lancez l'exécution de votre programme directement à partir de l'option 2 de la page de présentation.

2 UN PROGRAMME ENREGISTRE

En effet celle-ci n'est pas effacée par le TO7.

COLOR: Voir chapitre POSSIBILITÉS GRAPHIQUES

CONSOLE: Instruction

CONSOLE lp, ld, c, s

lp, ld numéros de ligne compris entre 0 et 24
c mode de progression des couleurs c = 0 ou 1
s mode de défilement s = 0, 1 ou 2.

L'instruction CONSOLE permet de se définir une zone de travail sur l'écran ; lp indique la première ligne de la zone de travail, ld la dernière.

Exemples :

1) Dans l'exemple suivant nous réservons les quatre premières lignes de l'écran pour écrire un titre qui ne devra jamais être effacé.

```
10 CLS: SCREEN 3,4,4
20 PRINT: ATTRB 1,1
30 PRINT "   EXEMPLE CONSOLE"
40 CONSOLE 4,24
50 CLS: ATTRB 0,0
60 PRINT "PREMIERE LIGNE ZONE DE TRAVAIL"
70 LOCATE 0,12
80 PRINT "POUR EFFACER APPUYER SUR UNE TOUCHE"
90 E$ = INPUT$(1)
100 CLS
```

Vous remarquerez que le titre n'est jamais effacé. Toutefois il est toujours possible de venir écrire dans la zone réservée en utilisant l'instruction LOCATE. Ainsi soulignons en rouge le titre "EXEMPLE CONSOLE" avant d'effacer l'écran. Nous rajouterons donc les instructions suivantes.

```
95 LOCATE 0,3 : ATTRB 1,1 : COLOR 1
96 PRINT "   ==============="
97 ATTRB 0,0 : COLOR 3
```

Avant de relancer l'exécution de votre programme entrez la commande CONSOLE 0 de façon à redisposer de la totalité de l'écran.

2) Dans l'exemple suivant nous séparerons l'écran en deux zones en spécifiant pour chacune d'entre elles des couleurs différentes pour les caractères. Ainsi dans la première zone, les caractères seront de couleur jaune avec un fond bleu et dans la seconde ils seront de couleur verte avec un fond noir. Le tour de l'écran ne pouvant pas être partagé nous le mettrons en noir dans les deux cas (voir instruction SCREEN).

```
  5 REM DEFINITION PREMIERE ZONE
 1Ø CLS
 2Ø CONSOLE Ø,12
 3Ø SCREEN 3,4,Ø
 4Ø PRINT "PREMIERE ZONE"
 5Ø REM DEFINITION DEUXIEME ZONE
 6Ø CONSOLE 13,24
 7Ø CLS
 8Ø SCREEN 2,Ø,Ø
 9Ø PRINT "DEUXIEME ZONE"
1ØØ CONSOLE Ø,24
```

Le paramètre c de l'instruction CONSOLE définit le mode de progression des couleurs.

Si c est égal à zéro, alors tous les caractères seront imprimés dans la couleur courante et ce quelque soient les couleurs choisies pour la zone dans laquelle l'impression est réalisée.

Si c est égal à un tous les caractères seront imprimés dans la couleur de la zone correspondante quelque soit la couleur courante.

Exemple:

3) Reprenons le programme de l'exemple précédent. Modifions l'instruction 1ØØ comme suit

```
1ØØ CONSOLE Ø,24,C
```

et rajoutons les instructions suivantes.

```
 95 C = Ø
 96 COLOR 1,2 'couleur courante des caractères rouge sur fond vert
11Ø LOCATE Ø,3
12Ø PRINT "ESSAI CONSOLE AVEC C= " ; C
13Ø LOCATE Ø,2Ø
14Ø PRINT "ESSAI CONSOLE AVEC C = " ; C
```

Lancez l'exécution du programme. Changez la valeur de C: 95 C = 1 puis relancez l'exécution du programme. Cette fois-ci les textes sont affichés avec les couleurs spécifiées pour chaque zone d'écran.

Le paramètre s permet de fixer le mode de défilement. Celui-ci sera normal si s est égal à zéro, lent si s est égal à un et se fera par page si s est égal à deux.

Exemple:

4)Pour voir ces différents modes, reprenons l'exemple 2 en modifiant l'instruction 1ØØ.

```
1ØØ CONSOLE Ø,24,1,S
```

Rajoutons les instructions suivantes:

```
 95 S = Ø
11Ø FOR I = 1 TO 1ØØ
12Ø PRINT I
13Ø NEXT I
14Ø CONSOLE Ø,24,Ø,Ø
```

La dernière instruction CONSOLE remet votre écran dans son état habituel. Relancez l'exécution du programme en modifiant l'instruction 95 en:

```
      95 S = 1
puis  95 S = 2
```

Voir aussi chapitre POSSIBILITÉS GRAPHIQUES.

COS: Fonction

COS (x)

x est un nombre exprimé en radians.

La fonction COS calcule le cosinus d'un angle exprimé en radians.

Exemple:

```
1Ø CLS
2Ø PI = 3.14159
3Ø PRINT "ENTREZ UN ANGLE EN DEGRES"
4Ø PRINT "LE PROGRAMME LE CONVERTIRA EN RADIANS
   POUR CALCULER SON COSINUS"
5Ø INPUT "ANGLE" ; A
6Ø C = PI * A/18Ø
7Ø PRINT "LE COSINUS DE L'ANGLE" ; A ; "EST" ; COS (C)
```

CSNG : Fonction

CSNG (x)

x est une variable de type entier ou double précision.

La fonction CSNG convertit une variable de type entier ou double précision en une variable de type simple précision. Pour les variables double précision un arrondi est effectué.

Exemple :

```
10 CLS
20 PI# = 3.1415926 : A% = 5 : S# = 4.1234568
30 PRINT CSNG (PI#), PI#
40 PRINT
50 PRINT CSNG (A%), A%
60 PRINT
70 PRINT CSNG (S#), S#
```

CSRLIN : Fonction

CSRLIN

La fonction CSRLIN renvoie le numéro de la ligne courante.

Exemples :

1)
```
10 CLS
20 FOR I = 0 TO 15
30 PRINT "LIGNE" ; CSRLIN
40 NEXT
```

2) Généralement dans un programme lors de l'entrée de données au clavier, un test est effectué pour vérifier leur validité. Si celles-ci sont incorrectes, un message d'erreur est imprimé. Ceci est parfois gênant. La séquence d'instructions suivantes permet d'effacer le message d'erreur et de se repositionner à l'endroit de l'écran où la demande d'acquisition avait été faite.

```
1ØØ L = CSRLIN : C = POS(Ø)
11Ø INPUT "NOMBRE" ; N
12Ø ' TEST .... IF .... THEN 3ØØ          DONNÉES CORRECTES
13Ø COLOR 1 : ? "ERREUR ...." : COLOR 4 : ? "APPUYER SUR LA
    TOUCHE" ; : COLOR Ø,3 : ? "ENTREE" :
14Ø E$ = INPUT$(1)
15Ø COLOR 4,6 : LOCATE L,C : ? CHR$(24) ;
15Ø CONSOLE L + 1 : CLS : CONSOLE Ø
```

DATA: Instruction non exécutable

DATA C1, C2

C1, C2 sont des constantes numériques ou alphanumériques.

L'instruction DATA permet d'inclure des données dans un programme BASIC. Une même instruction DATA peut contenir des constantes numériques et alphanumériques. Ces dernières peuvent ne pas être comprises entre guillemets sauf si elles contiennent un séparateur.

Exemple:

```
1Ø CLS
2Ø READ A, B, C$
3Ø LOCATE A, B
4Ø PRINT C$
5Ø READ A$
6Ø PRINT A$
7Ø DATA 5, 12, ESSAI DE DATA
BØ DATA "ESSAI, AVEC, VIRGULES"
```

Remarque: Il ne faut pas insérer de commentaires dans la même ligne de programme qu'une instruction DATA.

Ainsi

```
1ØØ READ A$, B$
12Ø PRINT A$, B$
13Ø DATA EXEMPLE : ' Exemple DATA
14Ø DATA DONNEES
```

devra être écrit pour fonctionner correctement

```
1ØØ READ A$, B$
12Ø PRINT A$, B$
13Ø DATA EXEMPLE
135 ' Exemple DATA
14Ø DATA DONNEES
```

DEFGR$: Voir chapitre POSSIBILITÉS GRAPHIQUES (tome 2)

DEFUSR: Voir chapitre ACCÈS AU LANGAGE MACHINE

DIM: Instruction

DIM n_1 (i_1, i_2, ...), n_2 (i_3, i_4, ...) ...

n_1, n_2 ...: noms de tableau

i_1, i_2, i_3, i_4 ...: nombre d'éléments de chaque dimension du tableau.

L'instruction DIM réserve la place en mémoire pour tous les éléments des tableaux n_1, n_2, ...

— Le nombre de dimensions est limité à 255.

— Les indices peuvent varier entre Ø et 32 767.

— Si un tableau de dimension inférieure ou égal à 3, possède moins de 11 éléments, (indice variant de Ø à 1Ø), alors il n'est pas nécessaire de le dimensionner.

— Les indices sont toujours arrondis au nombre entier le plus proche.

Exemple:

```
1Ø CLS
2Ø DIM B$ (5Ø)
3Ø REM TABLEAU C NON DIMENSIONNE
4Ø FOR I = Ø TO 7
5Ø FOR J = Ø TO 6
6Ø FOR K = Ø TO 1Ø
7Ø C (I, J, K) = I + J * K
8Ø NEXT K, J, I
```

DEFINT, DEFSNG, DEFDBL, DEFSTR

DEFINT
DEFSNG
DEFDBL
DEFSTR
} $a_1 - a_2, a_3, \ldots$

a_1, a_2, a_3 sont des lettres de l'alphabet.

Ces instructions permettent de définir implicitement un type pour les variables dont le nom commence :

— par la lettre de l'alphabet a_3.

ou

— par une lettre de l'alphabet placée entre a_1 et a_2.

DEFINT	définit des variables entières
DEFSNG	définit des variables réelles simple précision
DEFDBL	définit des variables réelles double précision
DEFSTR	définit des chaînes de caractères

Si un type de variable est précisé directement dans le nom alors la déclaration implicite pour cette variable est ignorée.

Exemple :

```
10 CLS
20 DEFINT A-C
30 DEFSTR S
40 DEFDBL H
50 A = 3.14 : B = 4.55 : C = 2.04
60 C != 1.47
70 SEC="CHAINE"
80 H3=2.7182818281828
90 PRINT A, B, C,
100 PRINT C !, SEC, H3
```

Les variables dont le nom commence par A, B, C sont de type entier, celles commençant par S de type chaîne et celles commençant par H réel double précision.

Toutes les autres sont des variables réelles simple précision.

Remarques :

1) Avec ces instructions vous n'êtes plus obligés de spécifier le caractère #, $ et % pour les variables double précision, alphanumériques et entières. Ceci peut procurer un gain de place important en mémoire.

2) Déclarer toujours le type de vos variables avant de les initialiser sinon elles seront remises à zéro.

END : Instruction

END

L'instruction END arrête l'exécution d'un programme et referme tous les fichiers ouverts.

L'exécution du programme peut être reprise par la commande CONT.

EOF : Fonction

EOF (x)

x est un numéro de canal.

La fonction EOF permet de détecter la fin d'un fichier séquentiel lors d'une relecture du fichier.

La valeur retournée sera – 1 si la fin du fichier est détectée et Ø sinon.

Exemple :

```
1Ø CLS
2Ø OPEN "I", #2, "DONNEES"
3Ø IF EOF (2) THEN 7Ø
4Ø INPUT #2, I
5Ø PRINT I
6Ø GOTO 3Ø
7Ø CLOSE 2     'détection de la fin du fichier
8Ø PRINT "FIN DU FICHIER"
```

ERL, ERR

ERL et ERR sont des variables gérées par le BASIC TO7 lorsqu'une erreur est détectée dans un programme. Ces deux variables mémorisent respectivement le numéro de la ligne programme en cours d'exécution au moment de l'erreur et le numéro de code de l'erreur. Utilisées avec une instruction ON ERROR GOTO, ces variables serviront à gérer certains types d'erreur lors de l'exécution d'un programme.

Voir aussi ON ERROR GOTO

ERROR: Instruction

ERROR n
n est un numéro d'erreur inférieur à 67.

L'instruction ERROR permet de simuler l'erreur dont le code est n. Elle provoque de ce fait l'arrêt de l'exécution du programme (sauf si une séquence de traitement d'erreur a été incluse dans le programme).

Exemple: Le programme suivant est syntaxiquement correct. Cependant quand vous lancez son exécution une erreur de syntaxe est détectée!

```
1Ø CLS
2Ø PRINT "ESSAI DE L'INSTRUCTION ERROR"
3Ø ERROR 2
4Ø PRINT "FIN DU PROGRAMME"
```

EXEC: Voir chapitre ACCÈS AU LANGAGE MACHINE.

EXP: Fonction

EXP (x)
x est un nombre décimal.

La fonction EXP permet de calculer l'exponentielle de x.

Exemple:

```
1Ø CLS
2Ø INPUT "X" ; X
3Ø PRINT "L'EXPONENTIELLE DE" ; X ; "EST" ; EXP (X)
```

FIX: Fonction

FIX (x)
x est un nombre décimal.

La fonction FIX renvoie la valeur entière de x par troncature. (Elle supprime donc la partie décimale de x).

Ainsi

FIX (1.95) = FIX (1.Ø1) = 1
et FIX (– 2.7) = FIX (– 2.4.) = – 2

FOR: Instruction

FOR I = XI TO XF STEP XP
I : variable de contrôle de la boucle
XI : valeur initiale de I
XF : valeur finale de I
XP : pas de progression de la boucle.

L'instruction FOR utilisée avec l'instruction NEXT permet la création d'une boucle. La variable de contrôle de la boucle I ne pourra pas être du type réel double précision. Le pas de progression XP peut être négatif.

Les boucles imbriquées sont autorisées, le niveau d'imbrication n'étant limité que par la taille mémoire.

Si la valeur initiale est supérieure à la valeur finale avec un pas de progression positif alors la boucle ne sera pas exécutée *mais la variable de contrôle de la boucle prendra toutefois la valeur initiale indiquée.*

Exemples:

1) 1ØØ FOR A = 5 + C TO C * B / D STEP – 2.5

2)
```
1Ø DIM B(1ØØ)
2Ø FOR I = Ø TO 1ØØ
3Ø B(I) = I
4Ø NEXT I
```

Remarques :

1) Si l'on "sort" de la boucle par l'instruction NEXT, la valeur de la variable de contrôle sera égale à la valeur finale augmentée du pas de progression. Ainsi dans l'exemple suivant, la valeur affichée pour I sera 9.

```
1Ø FOR I = 1 TO 2 STEP 8
2Ø NEXT I
3Ø PRINT I
```

2) Il est possible de sortir d'une boucle sans "passer" par l'instruction NEXT. Toutefois ceci présente **deux inconvénients.**

— La variable de contrôle de boucle est toujours considérée comme étant utilisée; sa place en mémoire est donc conservée. Ceci occupera 15 octets.

— Si la boucle dont on sort est une boucle imbriquée il faudra préciser le nom de la variable de contrôle de la boucle dans l'instruction NEXT sinon une erreur NF se produira. Ceci peut être vérifié sur l'exemple suivant.

```
1Ø CLS
2Ø FOR I = 1 TO 3
3Ø PRINT "AAA"
4Ø FOR J = 1 TO 2
45 PRINT "BBB"
5Ø GOTO 7Ø
6Ø NEXT
7Ø NEXT
```

modifiez les instructions 6Ø et 7Ø comme suit :

```
6Ø NEXT J
7Ø NEXT I
```

Le programme s'exécutera alors parfaitement.

3) *Voir aussi instruction NEXT*

FRE: Fonction

FRE (Ø)
FRE (x)
x est une variable alphanumérique.

La fonction FRE permet de connaître la place libre en mémoire.

FRE (Ø) donnera la place restante pour le programme et les variables.

FRE (A$) donnera la place restante pour les chaînes de caractères.

Exemple:

Dans sa version de base le TO7 dispose de 8KRAM en mémoire centrale. Regardons la place occupée par les variables internes du BASIC. Pour cela il suffit de rentrer en mode commande, dès la mise sous tension.

```
PRINT 8 * 1 Ø24 – FRE (Ø)
```

Ce qui donne 1 851 octets. Les variables du BASIC occupent donc environ 1,5 K en mémoire RAM. Parmi ces 1 851 octets 3ØØ sont réservés pour le traitement des chaînes de caractères. En effet entrez la commande CLEAR Ø. Ceci réservera zéro octets pour le traitement des chaînes de caractères. Si l'on redemande le nombre d'octets pris par le BASIC nous obtiendrons alors 1 551.

La commande PRINT FRE(A$) nous permettra de vérifier que le nombre d'octets réservés pour le traitement des chaînes de caractères est bien nul (suite à la commande CLEAR Ø).

GOSUB: Instruction

GOSUB n
n est un numéro de ligne.

L'instruction GOSUB effectue le branchement à un sous-programme dont la première instruction a le numéro n. Un sous-programme peut en appeler un autre, le nombre de niveaux d'appels ne dépend que de la taille mémoire disponible. En effet chaque fois qu'un

appel de sous-programme est effectué le BASIC TO7 conserve en mémoire l'adresse de retour ce qui occupe à chaque fois 7 octets en mémoire centrale. Ces sept octets seront "libérés" au retour du sous-programme. Ainsi un trop grand nombre d'appels "imbriqués" pourra utiliser toute la mémoire disponible.

Les sous-programmes recursifs sont autorisés (sous-programme qui s'appelle lui-même).

Exemples:

1)
```
 3ØØ GOSUB 1ØØØ
1ØØØ REM SOUS-PROGRAMME
11ØØ RETURN
```

2) L'exemple suivant permet de vérifier que l'appel d'un sous-programme prend sept octets en mémoire centrale.

```
 1Ø CLS
 2Ø PRINT "PROGRAMME PRINCIPAL"
 3Ø PRINT "PLACE DISPONIBLE"; FRE (Ø)
 4Ø GOSUB 1ØØ
 5Ø PRINT "PROGRAMME PRINCIPAL - RETOUR"
 6Ø PRINT "PLACE DISPONIBLE" ; FRE (Ø)
 7Ø END
1ØØ REM "SOUS-PROGRAMME"
11Ø PRINT "SOUS-PROGRAMME"
12Ø PRINT "PLACE DISPONIBLE" ; FRE (Ø)
13Ø RETURN
```

Remarque: *Voir aussi l'instruction RETURN.*

GOTO: Instruction

GOTO nl

n est un numéro de ligne.

L'instruction GOTO provoque un branchement à la première instruction de la ligne de programme n.

Exemple:

```
1ØØ GOTO 3ØØ
```

HEX$: Fonction

HEX$ (x)

x est une variable numérique dont la valeur est comprise entre Ø ET 65 535.

La fonction HEX$ donne la valeur hexadécimale d'une variable numérique.

Exemple: Le programme suivant donne la valeur hexadécimale d'un nombre entré au clavier.

```
 5 REM Conversion en hexadécimal
1Ø SCREEN 2,Ø,Ø
2Ø CLS: LOCATE Ø,Ø,Ø
3Ø INPUT "NOMBRE " ; N%
4Ø PRINT N% ; "EN DECIMAL EST EQUIVALENT A " ; HEX$ (N%) ;
" EN HEXADECIMAL"
5Ø INPUT "VOULEZ-VOUS CONTINUER (O/N) " ; R$
6Ø IF R$ "O" THEN GOTO 2Ø ELSE SCREEN 4,6,6
7Ø END
```

Remarque: Contrairement aux autres fonctions, si x n'est pas entier, la valeur prise pour la conversion sera celle du *nombre entier immédiatement inférieur* et non celle du nombre entier le plus proche.

Ainsi

```
PRINT HEX$ (1Ø.9) affichera A
```

IF ... THEN ... ELSE: Instruction

IF cc. THEN instructions ELSE instructions

cc est une condition

instructions une ou plusieurs instructions séparées par le caractère :

Si la condition *cc* est réalisée alors les instructions situées après THEN sont exécutées. Si la condition *cc* n'est pas réalisée, ce sont les instructions situées après ELSE qui seront exécutées. Si, dans ce deuxième cas, il n'y a rien à faire, le mot clé ELSE peut être supprimé Toutes les instructions, même une nouvelle instruction IF, peuvent être utilisées après THEN et ELSE.

Quand plusieurs instructions IF sont imbriquées, chaque ELSE rencontré est associé au premier THEN précédent "libre" (c'est-à-dire qui n'est déjà pas, lui-même, associé à un ELSE).

Exemples :

1)
```
1ØØØ IF A$ = "NON" THEN END ELSE GOTO 1Ø
```

2)
```
1Ø CLS
2Ø INPUT "JOUR, MOIS" ; J, M
3Ø IF J >= Ø AND J =< 7 THEN IF M <Ø OR M> 12 THEN
   PRINT "MOIS INCORRECT" ELSE PRINT "DATE
   CORRECTE" ELSE IF M >Ø AND M < 13 THEN PRINT
   "JOUR INCORRECT" ELSE PRINT "JOUR ET MOIS
   INCORRECTS"
```

Dans cet exemple, l'instruction IF est évaluée comme suit.

```
IF J >= Ø AND J =< 7
   ┌→ THEN IF M < Ø OR M > 12
   │              ┌→ THEN PRINT "MOIS INCORRECT"
   │              └→ ELSE PRINT "DATE CORRECTE"
   └→ ELSE IF M > Ø AND M < 13
                  ┌→ THEN PRINT "JOUR INCORRECT"
                  └→ ELSE PRINT "JOUR ET MOIS INCORRECTS"
```

Les flèches montrent les associations des mots clés THEN et ELSE.

Remarque :

1) Reprenons l'exemple 2 en supposant que l'on ne désire rien imprimer si la date est correcte. Il suffit pour cela de supprimer l'instruction PRINT après ELSE dans le IF imbriqué. Toutefois pour garder une association correcte entre les mots clés THEN et ELSE, il faudra *obligatoirement* laisser le ELSE. Examinons les associations réalisées dans les deux cas.

1) en laissant le ELSE

```
IF J>=Ø AND J=<7
   ┌→THEN IF M<Ø OR M>12
   │            ┌→ THEN PRINT "MOIS INCORRECT"
   │            └→ ELSE
   └→ELSE IF M>Ø AND M<13
                ┌→ THEN PRINT "JOUR INCORRECT"
                └→ ELSE PRINT "JOUR ET MOIS INCORRECTS"
```

Ce qui est bien ce que nous cherchons.

2) en supprimant le ELSE

```
IF J> = Ø AND J = < 7
      THEN IF M <Ø OR M> 12

          ┌→ THEN PRINT "MOIS INCORRECT"
          └→ ELSE  IF M>Ø AND M<13

                ┌→ THEN PRINT "JOUR INCORRECT"
                └→ ELSE PRINT "JOUR ET MOIS INCORRECTS"
```

3) Lorsque le BASIC TO7 évalue une expression logique il affecte à une variable interne la valeur — 1 si l'expression est vraie, et Ø si l'expression est fausse. On pourra donc tester très simplement si une valeur est différente de zéro de la façon suivante.

```
écrire     1ØØ IF A THEN PRINT "A <> Ø" ELSE PRINT "A = Ø"
au lieu de 1ØØ IF A = Ø THEN PRINT "A = Ø" ELSE PRINT
           "A < > Ø"
```

Ceci sera interprété ainsi :

Si A est différent de zéro la condition est vraie (même si A est différent de — 1).

Si A = Ø la condition est fausse.

INKEY$: Fonction

INKEY$

La fonction INKEY$ permet de saisir "au vol" le caractère correspondant à la touche pressée au clavier.

Exemples:

1)
```
10 CLS
20 PRINT "APPUYER SUR UNE TOUCHE . POUR ARRETER,
   APPUYEZ SUR LA TOUCHE F"
30 TOUCHE$ = INKEY$
40 IF T$ = "F" THEN PRINT "FIN" : END ELSE LOCATE 0, 2, 0 :
   PRINT "VOUS AVEZ APPUYE SUR LA TOUCHE" ; TOUCHE$
   : GOTO 30
```

2) Le petit programme suivant permet de réaliser un dessin sur l'écran en utilisant les touches de déplacement du curseur. Les touches RAZ , L, P et E sont respectivement utilisées pour effacer l'écran (le dessin reprendra alors à partir du dernier point tracé), arrêter un trait continu, reprendre un trait continu et stopper l'exécution du programme. La touche R sélectionne le mode répétition. Toutes vos commandes seront par la suite répétées. La touche S permet de revenir au mode pas à pas.

```
10 CLS : SCREEN 3,4,4 : LOCATE 0,0,0 : PSET (160, 100)
20 X = 160 : Y = 100 : M$ = "S" : C$ = CHR$ (9)
30 T$ = INKEY$
35 IF T$ = " " THEN IF M$ = "R" THEN T$ = C$ ELSE 30
40 IF ASC (T$) = "12" THEN COLOR 3 : CLS : PSET (X,Y) :
   GOTO 30
50 IF T$ = "L" THEN LC = 1 : I = 1 ELSE IF T$ = "P" THEN
   LC = 0 : GOTO 30
55 IF LC = 1 AND I = 0 THEN PSET (X,Y),   - 5
60 IF ASC (T$) =  9 THEN X = X + 1 : GOTO 100
70 IF ASC (T$) =  8 THEN X = X - 1 : GOTO 100
80 IF ASC (T$) = 10 THEN Y = Y + 1 : GOTO 100
90 IF ASC (T$) = 11 THEN Y = Y - 1
100 IF X < 0 THEN X = 0
110 IF Y < 0 THEN Y = 0
120 IF T$ <> "R" AND T$ <> "L" THEN C$=T$
125 IF "T$" = "R" THEN M$ = "R"
126 IF "T$" = "S" THEN M$ = "S"
130 PSET (X,Y) : I = 0
140 GOTO 30
```

Remarques :

1) En réalité la fonction INKEY$ n'effectue pas une véritable scrutation du clavier mais vient simplement lire le contenu d'un registre interne au TO7 dans lequel est mémorisé la dernière touche pressée pendant que le TO7 exécutait un programme, un listage, une sauvegarde... Nous pouvons le vérifier avec le programme suivant dont les instructions 2 à 4Ø sont des instructions REM.

```
 1 CLS
 2 REM
 3 REM
   |
38 REM
39 REM
4Ø REM
5Ø PRINT "TOUCHE APPUYEE :" ; INKEY$
```

Demandons un listage de ce programme et arrêtons le en appuyant sur la touche STOP. Pressons la touche L pour faire reprendre le listage puis lançons l'exécution du programme. Nous obtenons le message TOUCHE APPUYEE : L.

Ceci se révèle parfois gênant. Pour éliminer ce caractère restant il suffira alors d'exécuter au moins une fois la fonction INKEY$ avant le cas à traiter.

Dans le programme précédent l'instruction 1 deviendrait.

```
1) CLS : A$ = INKEY$
```

2) Après avoir lu le registre contenant la dernière touche appuyée, la fonction INKEY$ remet le contenu de celui-ci à zéro. Ainsi si un appui touche possède plusieurs significations, il faudra le mémoriser dans une variable alphanumérique.

Exemple : Considérons le petit programme suivant écrit de deux façons différentes.

1)
```
1Ø CLS
2Ø IF INKEY$ = "A" THEN PRINT "A" ;
3Ø IF INKEY$ = "B" THEN PRINT "B" ;
4Ø GOTO 2Ø
```

2) 1Ø CLS
2Ø T$ = INKEY$
3Ø IF T$ = "A" THEN PRINT "A" ;
4Ø IF T$ = "B" THEN PRINT "B" ;
5Ø GOTO 2Ø

Ce programme teste si vous appuyez sur la touche A ou B du clavier et l'affiche. Toutes les autres touches sont ignorées. Lançons l'exécution du programme 1 et appuyons sur les touches A et B. Nous constatons parfois des "ratés" c'est-à-dire nous entendons bien le "beep" indiquant que l'appui touche est valide mais la lettre correspondante n'est pas affichée. Recommençons la même opération avec le second programme. Cette fois ci il n'y a plus aucun problèmes! Pourquoi?

Reprenons le premier programme et supposons que lorsque vous commencez à appuyer sur la touche A, le TO7 exécute l'instruction 2Ø. La touche A ne sera donc pas reconnue. A la fin de l'exécution de l'instruction 2Ø, le TO7 détectera l'appui sur la touche A, enverra un "beep" pour signaler que l'appui-touche est valide, mémorisera dans son registre interne la lettre A et passera enfin à l'exécution de l'instruction 3Ø. La fonction INKEY$ lira le contenu du registre interne et remettra celui-ci à vide. Ainsi lorsque l'instruction 2Ø est réexécutée, le registre interne ne contient plus la touche "A" mais est initialisé à vide.

Avec le second programme, le résultat de la fonction INKEY$ est mémorisé dans une variable alphanumérique qui pourra être référencée autant de fois que nécessaire ce qui ne réinitialise pas son contenu.

3) *N'utilisez pas la touche STOP avec INKEY$.* En effet STOP arrête l'exécution du programme.

INPEN: Voir chapitre CRAYON OPTIQUE.

INPUT "texte" $\left\{ \begin{matrix} ; \\ , \end{matrix} \right\}$ x_1, x_2,...

texte est une chaîne de caractère.

x_1, x_2 ... sont des noms de variables.

L'instruction INPUT permet l'entrée de données lors de l'exécution du programme. Le texte entre guillemets sera affiché suivi d'un point

d'interrogation et d'un espace si la liste des variables est précédée par un point virgule. Si la liste des variables est précédée par une virgule, seul le texte entre guillemets sera affiché. Les caractères que vous entrerez seront alors collés à ce texte.

Exemple:

```
10 COLOR 2,0 : SCREEN,, 0 : CLS
20 PRINT "INPUT AVEC UN POINT VIRGULE"
30 X=59 : GOSUB 200
40 INPUT "JOUR" ; M$ : ? : ? :
50 PRINT "INPUT AVEC VIRGULE"
60 X=44 : GOSUB 200
70 INPUT "JOUR", M$
80 END
200 PRINT "INPUT" ; CHR$(34) ; "texte" ; CHR$(34) ;
210 COLOR 1 : PRINT CHR$ (X) ; : COLOR 2 : PRINT "variables, ...";
    SPC (5) ; "donne" : ? : ?
220 RETURN
```

Remarque: Si l'on désire rentrer plusieurs variables avec une seule instruction INPUT, il faudra rentrer leurs valeurs en une seule fois en séparant chacune d'entre elles par une virgule.

Exemple:

```
10 CLS
20 INPUT "DATE (JJ/MM/AA)" ; J, M, A
```

quand le message DATE (JJ,MM,AA) ? apparaîtra sur l'écran vous entrerez au clavier 01,10,83 ENTREE .

INPUT#: Instruction

INPUT# c, x_1, x_2 ...

c est un numéro de canal

x_1, x_2 sont des noms de variables.

L'instruction INPUT# permet de relire des données stockées sur un fichier.

Exemple:

```
1Ø CLS
2Ø OPEN "I", 1, "DONNEE.DAT"
3Ø PRINT "DEBUT LECTURE"
4Ø IF EOF(1) THEN 7Ø
5Ø INPUT#1, I
6Ø PRINT I ; : GOTO 4Ø
7Ø PRINT "FIN LECTURE"
8Ø CLOSE : END
```

INPUT$: Fonction

INPUT$ (x)

x nombre décimal compris entre Ø ET 255.

La fonction INPUT$ attend que l'utilisateur appui sur x touches du clavier. Tous les appuis touche sont mémorisés. L'exécution du programme reprendra dès que le nombre de caractères attendus aura été rentré.

Exemple:

```
1Ø CLS
2Ø PRINT "FAITES TROIS APPUI-TOUCHE"
3Ø A$ = INPUT$(3)
4Ø PRINT "VOUS AVEZ APPUYE SUR LES TOUCHES" ; A$
```

INPUTPEN: Voir chapitre CRAYON OPTIQUE

INPUTWAIT: Instruction

INPUTWAIT n ; t, "texte" { ; , } x_1, x_2

n est un numéro de ligne
t est un temps en secondes
texte est une chaîne de caractères
x_1, x_2... sont des noms de variables

L'instruction INPUTWAIT permet l'introduction de données au clavier dans un temps limité. Si l'utilisateur n'a pas entré ses données au

bout du temps t, l'exécution du programme continuera à partir de l'instruction n.

Exemple:

```
1Ø CLS : SCREEN 2, Ø, Ø
2Ø INPUTWAIT 3Ø ; 5, "ATTENDEZ 5 SECONDES"
3Ø BEEP : PRINT "APPUYEZ SUR UNE TOUCHE EN MOINS DE
   DEUX SECONDES SINON LE PROGRAMME S'AUTO-DETRUI-
   RA !..."
4Ø INPUTWAIT 7Ø ; 2, "DEPECHEZ-VOUS !", A$
5Ø FOR I = 1 TO 2Ø : PRINT CHR$(7) ; : NEXT
6Ø LOCATE 15,13,Ø : ATTRB 1,1 : COLOR 1 : PRINT "BRAVO" :
   COLOR 2 : END
7Ø ATTRB 1,1 : LOCATE 1Ø,13,Ø : PRINT "TROP TARD !" :
   ATTRB Ø,Ø : NEW
```

Remarque : Si vous faîtes une erreur lorsque vous entrez vos variables, le chronométrage recommencera à partir de zéro seconde.

INSTR : Fonction

INSTR (i, chaîne 1, chaîne 2)

i nombre compris entre 1 et 255, peut être omis
chaîne 1, chaîne 2: chaînes de caractère.

La fonction INSTR renvoie la position de la chaîne 2 dans la chaîne 1. Si i est spécifié la recherche commencera à partir de la i^e lettre de la chaîne 1. Si i est omis, la recherche commencera au début de la chaîne 1. Si i a une valeur supérieure au nombre de caractères de chaîne 1, la valeur retournée par la fonction INSTR sera nulle.

Exemple: Le programme suivant recherchera une configuration de caractères spécifiés dans une chaîne et l'affichera en rouge. Pour arrêter le programme appuyer sur la touche S.

```
10 CLS
20 CONSOLE Ø, 12
30 LOCATE Ø,6,Ø : T$ = "JE SUIS LE TØ7 DE THOMSON ET
   NOUS VOYONS L'INSTRUCTION INSTR"
40 CLS : PRINT T$
50 INPUT "CHAINE RECHERCHEE" ; N$
60 INPUT "A PARTIR DU CARACTERE NUMERO (ESPACES
   INCLUS)" ; D
70 IF D < 1 OR D>255 THEN 6Ø ELSE IF D>LEN(T$) THEN
   PRINT "LA POSITION DE DEPART DEPASSE LA CHAINE"
80 A = INSTR (D, T$, N$)
90 IF A = Ø THEN PRINT "LA CHAINE DE CARACTERES " ; N$ ;
   "N'EXISTE PAS" : GOTO 12Ø
100 LOCATE Ø,6,Ø : ? LEFT$ (T$, A – 1) ; : COLOR 1 : PRINT N$ ; :
   COLOR 2 : PRINT RIGHT$ (T$, LEN (T$) – A – LEN (N$) + 1)
120 LOCATE Ø, COLOR 1 : PRINT "APPUYER SUR UNE TOUCHE"
   ; : COLOR 2 : (E$) = INPUT$(1) : IF E$ = "S" THEN END
   ELSE 3Ø
```

INT : Fonction

INT (x)

x est une variable numérique.

La fonction INT renvoie la partie entière du nombre x c'est-à-dire le premier nombre entier strictement inférieur ou égal à x.

Exemple:

```
10 CLS
20 PRINT INT (4.5), INT (– 4.5)
30 PRINT INT (4), INT (– 4)
```

LEFT$: Fonction

LEFT$ (c, i)

c est une chaîne de caractères

i est un nombre compris entre Ø et 255

La fonction LEFT$ renvoie les i premiers caractères de la chaîne c.

Exemple:

```
10 CLS : SCREEN 2,0,0
20 INPUT "ENTRER VOTRE TEXTE"; T$
30 INPUT "NOMBRE DE CARACTERES A PRENDRE" ; N
40 PRINT "LES" ; N ; "PREMIERS CARACTERES DU TEXTE" ;
   T$ ; "SONT : " ;
50 COLOR 1
60 PRINT LEFT$ (T$, N) : COLOR 2
70 INPUT "FIN (O/N)" ; R$
80 R$ = "O" THEN END
90 INPUT "NOUVEAU TEXTE (O/N)" ; R$
100 CLS : IF R$ = "O" THEN 20 ELSE 30
```

LEN: Fonction

LEN (c)

c est variable alphanumérique.

La fonction LEN renvoie le nombre de caractères de la variable alphanumérique c (c'est-à-dire sa longueur). Une longueur nulle signifiera que la variable a été initialisée à vide.

Exemple: Le programme suivant affichera la moitié d'un texte.

```
10 CLS
20 INPUT "TEXTE" ; T$
30 PRINT LEFT$ (T$, LEN(T$) / 2)
```

LET: Instruction

LET x = y

x est un nom de variable (numérique ou alphanumérique)

y est une expression numérique ou une chaîne de caractères ou un nom de variable.

L'instruction LET affecte à la variable x, la valeur de l'expression y. Le mot-clé LET est facultatif.

Exemple:

```
1Ø CLS
2Ø LET A = 15
3Ø B = – 5
4Ø PRINT A, B
```

Remarque: La forme particulière suivante de l'instruction LET permet d'affecter à une variable le résultat d'un test.

Exemple:

```
1Ø LET X = expression logique
1Ø LET X = 3 * Y < 9
```

Ceci sera interprété par le TO7 de la façon suivante.

si 3 * y est inférieur à 9 alors donner la valeur – 1 à X.

si 3 * y est supérieur ou égal à 9 alors donner la valeur Ø à X.

LINE: Voir chapitre POSSIBILITES GRAPHIQUES

LINE INPUT: Instruction

LINE INPUT "texte" { ; , } x

texte est un message qui sera affiché à l'écran
x est une variable alphanumérique

L'instruction LINE INPUT permet l'acquisition d'une chaîne de caractères comportant des signes de ponctuation, des guillemets, etc...

Cette instruction est analogue à l'instruction

INPUT "texte" { ; , } variable alphanumérique à la différence près que nous ne mettrons pas de guillemets si le texte à introduire comporte des virgules.

Exemple:

```
1Ø CLS
2Ø LINE INPUT "TEXTE" ; T$
```

LINE INPUT#: Instruction

LINE INPUT# n, c

n est un numéro de canal
c est une variable alphanumérique

Cette instruction est analogue à l'instruction LINE INPUT. Toutefois l'acquisition de la chaîne de caractères ne se fera plus au clavier mais à partir du fichier affecté au canal de numéro n.

Exemple: Ce petit programme fera l'acquisition des enregistrements du fichier FIC.

```
1Ø CLS
2Ø OPEN "I", #4, "FIC"
3Ø IF EØF(4) THEN CLOSE : END
4Ø LINE INPUT #4, ENR$
5Ø PRINT ENR$
6Ø GOTO 3Ø
```

LOCATE: Voir chapitre POSSIBILITÉS GRAPHIQUES.

LOG: Fonction

LOG (x)

x est une variable numérique

La fonction LOG calcul le logarithme népérien de x.

Exemple:

1) Le programme suivant permet de tracer la fonction logarithme népérien entre Ø.1 et 3Ø.

```
1Ø CLS: LOCATE Ø,Ø,Ø : SCREEN 2,Ø,Ø
2Ø LINE (Ø,1Ø2) – (32Ø,1Ø2), – 2
3Ø LINE (7,Ø) – (7,2ØØ), 1
4Ø FOR I = Ø.1 TO 3Ø STEP Ø.1
5Ø PSET (I * 1Ø + 7, 1Ø2 – 3Ø * LOG (I))
6Ø NEXT
7Ø END
```

2) Le programme suivant calcule le logarithme népérien d'un nombre quelconque.

```
10 CLS :SCREEN 2,0,0
20 INPUT "NOMBRE";N
30 IF N =<0 THEN COLOR 1:PRINT "N DOIT E
TRE STRICTEMENT POSITIF": COLOR 2  GOTO
 20
40 PRINT "LE LOGARITHME NEPERIEN DE";N."
EST:";LOG(N)
50 INPUT "ARRET (O/N)";REP$
60 IF REP$="O" THEN END ELSE 10
```

MID$: Instruction

MID$ (c_1, n_1, n_2) = c_2

c_1 et c_2 sont des chaînes de caractères

n_1 et n_2 sont des variables numériques (n_2 est facultatif)

n_1 doit être inférieur ou égal à la longueur de la chaîne c_1.

L'instruction MID$ permet de remplacer n_2 caractères de la chaîne c_1 par n_2 caractères de la chaîne c_2. Le remplacement dans la chaîne c_1 s'effectuera à partir du n_1 ème caractère.

La longueur de la chaîne c_1 restera fixe:

Si n_2 est omis, la totalité de la chaîne c_2 sera pris en compte.

Si n_2 est inférieur à LEN (c_2) alors seuls les n_2 premiers caractères de c_2 seront considérés.

Si n_2 est supérieur à LEN (c_2) alors toute la chaîne c_2 sera considérée.

Exemple:

1)
```
10 CLS
20 A$="ABCDEF"
30 MID$(A$,4,2)="XY"
40 PRINT A$
```

2)
```
10 CLS
20 A$="ABCDEFGH":B$="VWXYZ":C$=A$
30 MID$(A$,4,1)=B$
40 MID$(C$,2,15)=B$
50 PRINT A$,C$
```

3)
```
10 CLS
20 A$ = "CHAINE UN" : B$" DEUX PLUS LONGUE"
30 MID$ (A$, 8, 16) = B$
40 PRINT A$
```

Avec ce petit exemple vous pouvez constater que la chaîne de caractères **A$** a gardé la même longueur. Les caractères supplémentaires de la chaîne **B$** ont été ignorés.

4) Le programme ci-après simule l'instruction **MID$**. Les caractères que vous pensez remplacer apparaîtront en jaune sur fond bleu (après clignotement). Les caractères effectivement remplacés dans la chaîne apparaîtront en rouge sur fond blanc.

```
10 CLS : V$=""
20 INPUT "ENTRER UNE CHAINE DE 40 CARACT
ERES MAXIMUM";C$
30 IF LEN(C$)>40 THEN 10
40 CLS:PRINT C$
50 CONSOLE 12:CLS
55 IF V$<>"" THEN 70
60 INPUT"CHAINE REMPLACANTE";R$
70 INPUT"RANG DU CARACTERE A REMPLACER";
R
80 L=CSRLIN : C=0
90 IF R>LEN(C$) THEN COLOR 1:PRINT "LE R
ANG DU PREMIER CARACTERE A REMPLACER DOI
T ETRE INFERIEUR OU EGAL A LA LONGUEUR D
E LA CHAINE":COLOR 2:PRINT "APPUYER SUR
";:COLOR 0,3:PRINT "ENTREE":COLOR 2,0:E$
=INPUT$(1):CONSOLE L-1:CLS:CONSOLE 12:GO
TO 70
100 R=R-1:LOCATE R,0,0:COLOR 3,4:PRINTCH
R$(SCREEN(R,0));:COLOR 2,0
110 LOCATEC,L,1:INPUT"NOMBRE DE CARACTER
ES A REMPLACER";N
120 CLS:COLOR 3,4
130 E=R+N
```

```
140 IF E>LEN(C$) THEN E=LEN(C$)
150 FOR I=1 TO 8
160   COLOR ,,1:LOCATE R,0,0
170   FOR J=R TO E-1
180     PRINT CHR$(SCREEN(J,0));
190   NEXT
200   FOR X=1 TO 50:NEXT X
210 NEXT
220 LOCATE R,20,0:COLOR 1,0
230 IF R+LEN(R$)>40 THEN PRINT LEFT$(R$,
40-R);MID$(R$,41-R,LEN(R$)) ELSE PRINT R
$
240 IF R+N>LEN(C$) THEN N=LEN(C$)-R
250 COLOR 1,7:LOCATE R,20,0:M1=200:GOSUB
 510
260 M1=25:COLOR 1,0:LOCATE R,20,0:PRINT
LEFT$(R$,N)
270 COLOR 1,7:LOCATE R,19,0:GOSUB 510
280 FOR K=19 TO 1 STEP -1
290 COLOR 2,0:LOCATE R,K,0:PRINT SPC(N);
300 COLOR 1,7:LOCATE R,K-1,0:GOSUB 500
310 NEXT K
320 COLOR 2,0:CLS:INPUT"VOULEZ-VOUS CONT
INUER (O/N)";V$
330 CONSOLE 0:IF V$="N" THEN END
340 INPUT"VOULEZ-VOUS CHANGER DE CHAINE
DE CARACTERES (O/N) ";V$
350 IF V$="O" THEN 10 ELSE 40
500 PRINT CHR$(7);
510 PRINT LEFT$(R$,N)
520 FOR M=1 TO M1
530 NEXT M
540 RETURN
```

Remarque :

1) Ne confondez pas *l'instruction* **MID$** et *la fonction* **MID$** !

MID$: Fonction

MID$ (c, n_1, n_2)

c est une chaîne de caractères

n_1 expression numérique dont la valeur est comprise entre 1 et 255

n_2 expression numérique dont la valeur est comprise entre 1 et 255.

La *fonction* MID$ renvoie n_2 caractères de la chaîne de caractères c. Les caractères sont choisis à partir du n_1 ème caractère de la chaîne c. Si n_2 est omis, toute la chaîne c sera considérée à partir du rang n_1.

Exemple: Le petit programme suivant vous permettra d'essayer différentes combinaisons de la fonction MID$. Essayer en indiquant des valeurs nulles pour n_2 et des valeurs de n_1 supérieures à la longueur de votre chaîne de caractères.

```
10 CLS: SCREEN 2,0,0
20 INPUT "ENTREZ VOTRE CHAINE DE CARACTE
RES        ";R$
30 CONSOLE 10
40 CLS
50 INPUT "A PARTIR DE QUEL CARACTERE";N
60 INPUT "NOMBRE DE CARACTERES (ENTREE P
OUR N OMIS)";M$
70 IF M$<>"" THEN A$=MID$(R$,N,VAL(M$))
ELSE A$=MID$(R$,N)
80 PRINT "CHAINE ENTREE: ";R$
90 PRINT "LA CHAINE OBTENUE AVEC MID$(R$
,";N;
100 IF M$<>"" THEN PRINT ",";M$;
110 PRINT ")";"EST "; : COLOR 1  PRINT
A$ : COLOR 2
120 INPUT "ARRET (O/N)";T$
130 IF T$="O" THEN CONSOLE 0 :END
140 INPUT "NOUVELLE CHAINE (O/N)";T$
150 IF T$="O" THEN CONSOLE 0  GOTO 10 E
LSE 40
```

MOTOROFF
MOTORON } : Instructions

Mise en route et arrêt du magnétophone.

NEXT: Instruction

NEXT i_1, i_2,...

i_1, i_2 variables de contrôle de boucle. i est facultatif.

L'instruction NEXT incrémente la variable de contrôle de boucle du pas de progression. Le contrôle sera donné:

— à la première instruction après FOR si la boucle n'est pas terminée.

— à l'instruction suivante sinon.

Si des boucles imbriquées ont la même instruction finale il est possible d'utiliser une seule instruction NEXT avec les différentes variables de contrôle de boucle séparées par des virgules et en commençant par la boucle la plus imbriquée.

Exemple: Le programme suivant dessine un triangle rempli d'astérisques de couleur différente.

```
10 CLS:SCREEN 2,0,0
20 FOR I=0 TO 20
30   LOCATE 0,I,0
40   FOR J=0 TO I
50   COLOR (I+1+J) MOD 8
60   PRINT "*";
70 NEXT J,I
80 COLOR 2
```

Remarques:

1) La variable de contrôle de boucle après NEXT peut être omise. Le basic du TO7 considérera alors que ce NEXT se rapporte à la dernière instruction FOR rencontrée. Si plusieurs boucles imbriquées ont la même instruction NEXT (60 NEXT J,I exemple), elles devront être "éclatées" en plusieurs instructions NEXT. Ainsi l'instruction 70 de notre programme pourra être réécrite comme suit:

```
70 NEXT:NEXT
```

Ceci est avantageux lorsque vous avez besoin de vitesse. Une boucle sera exécutée beaucoup plus rapidement si vous omettez les variables de contrôle de boucle même si cela vous oblige à mettre

plusieurs instructions NEXT. Ceci peut être vérifié avec le petit programme classique suivant:

```
10 CLS
20 FOR I = 1 TO 10000
30 NEXT I
40 PLAY "DO" : PRINT I
```

Chronométrez le temps pris par l'exécution de ce programme (l'exécution sera terminée quand vous entendrez la note DO). Vous obtenez un temps compris entre 17 et 18 secondes.

Changez maintenant l'instruction 30 NEXT I en 30 NEXT et relancez l'exécution du programme vous obtenez un temps compris entre 12 et 13 secondes. 5 s de moins qu'auparavant! Ceci pourra être très utile pour réaliser des graphiques animés.

2) Voir aussi INSTRUCTION FOR.

OCT$: Fonction

OCT$ (x)

x est une variable numérique dont la valeur est comprise entre 0 ET 65 535.

La fonction OCT$ donne la valeur octale de la variable x.

Exemple: Le programme suivant affiche un nombre en hexadécimal. A vous de le convertir en octal!...

```
10 CLS : SCREEN 2,0,0
20 LOCATE 7,3,0 : PRINT"HEXADECIMAL ----
---> OCTAL"
30 BOX(48,19)-(8*34,35),1
35 CONSOLE 5
40 A=INT(RND*62534)
50 LOCATE 5,7 : PRINT "LE NOMBRE HEXADEC
IMAL EST: ";HEX$(A) : LOCATE 0,10
60 INPUT "ENTREZ VOTRE SOLUTION PREDEDEE
 DE &0 (EXEMPLE: &010=8) ",N
70 IF N=A THEN C$="BON":PLAY"T102DOMIS00
3DOMIS004DOMIS0" : X=19 ELSE C$="MAUVAI
S" : X=17: PLAY"T104SOMID003SOMID002SOMI
D0"
```

```
80 COLOR 3,4 : ATTRB 1,1
90 FOR I=1 TO 30
100   LOCATE X,20,0   PRINT C$
110   FOR J=1 TO 30 : NEXT J
120   COLOR ,,1
130 NEXT I
140 LOCATE X,20 : COLOR 2,0 : PRINT "
    " : ATTRB 0,0
150 IF C$="MAUVAIS" THEN PRINT "LA REPON
SE ETAIT :";:COLOR 1: PRINT "&O";OCT$(A)
 : COLOR 2
160 INPUT "CONTINUE (O/N) ";C$
170 IF C$="N" THEN CONSOLE 0 : END ELSE
CLS : GOTO 40
```

ON ERROR GOTO : Instruction

ON ERROR GOTO nl

nl : numéro de ligne du programme.

L'instruction ON ERROR GOTO provoque un débranchement au numéro de ligne nl si une erreur est détectée pendant l'exécution du programme. On pourra alors traiter cette erreur en utilisant les variables réservées du BASIC TO7 : ERR et ERL. La séquence de traitement de l'erreur devra se terminer par une instruction RESUME.

Exemple : Reprenons l'exemple de la fonction OCT$ et rajoutons l'instruction inexistante suivante 45 WWW pour créer une erreur de syntaxe. Suite à l'instruction 35 CONSOLE 5 nous ne disposons plus de la totalité de l'écran. Rajoutons donc une séquence de traitement d'erreur afin de pouvoir redisposer des six premières lignes de l'écran. Nous insérerons donc les instructions suivantes.

```
5 ON ERROR GOTO 1000

1000 CONSOLE 0
1010 RESUME NEXT
```

Une instruction ON ERROR GOTO nl est annulée par ON ERROR GOTO Ø. On veillera à ne pas introduire l'instruction ON ERROR GOTO Ø dans une séquence de traitement de l'erreur.

ON x GOTO nl_1, nl_2,...
ON x GOSUB nl_1, nl_2,...

x est une variable dont la valeur est comprise entre Ø et 255.

nl_1, nl_2 sont des numéros de lignes.

L'instruction ON x GOTO (ON x GOSUB) provoque un débranchement à la ligne programme nl_1 si $x = 1$, à la ligne programme nl_2 si $x = 2$,... (ou un appel du sous-programme dont la première instruction a le numéro de ligne nl_1, nl_2,...). Si la valeur de x est nulle ou supérieure au nombre de numéro de lignes présents dans l'instruction alors l'instruction suivante sera exécutée.

Exemple:

```
10 CLS : SCREEN 2,0,0
20 PRINT "ENTRER UN NOMBRE COMPRIS ENTRE
 1 ET 4 ET APPUYER SUR ";
30 COLOR 0,3   PRINT "ENTREE";: COLOR 2,
0
40 INPUT N
50 IF N < 0 OR N > 255 THEN PRINT "NOMBR
E INCORRECT (NEGATIF OU SUPERIEUR A 255)
" : GOTO 20
60 ON N GOTO 100,200,300,400
70 PRINT "NOMBRE NUL OU SUPERIEUR A 4"
 GOTO 20
100 PRINT " VOUS AVEZ ENTRE UN ":END
200 PRINT " VOUS AVEZ ENTRE DEUX ":END
300 PRINT " VOUS AVEZ ENTRE TROIS ":END
400 PRINT " VOUS AVEZ ENTRE QUATRE ":END
```

N
OPEN GOTO, ONPEN GOSUB

Voir chapitre UTILISATION DU CRAYON OPTIQUE.

OPEN: Instruction

OPEN "sens", # nc, "nom de fichier"

sens: indique le sens dans lequel les informations sont échangées

O pour OUTPUT (Écriture)
I pour INPUT (Lecture)
R ou D pour les fichiers à accès direct (disquette uniquement).

nc est un numéro de canal compris entre 1 et 15
"nom de fichier" est un nom de fichier.

Quand vous désirez lire ou écrire des données sur la cassette (ou sur la disquette), le BASIC TO7 utilise une mémoire tampon ("buffer") de 256 octets pour la cassette et de 128 octets pour la disquette. Chaque fois que vous demanderez une écriture ou une lecture, le TO7 remplira cette zone mémoire et l'écrira uniquement sur le périphérique correspondant lorsqu'elle sera pleine. Quand l'exécution de votre programme sera terminée, il est peu probable que cette mémoire tampon soit pleine. Il faudra alors rajouter une instruction CLOSE pour écrire les informations restantes. Le numéro de canal est un nombre compris entre 1 et 15 qui vous permet de "repérer" cette zone mémoire lorsque vous travaillez avec un ou plusieurs fichiers.

Exemple:

```
10 OPEN "O",#2,"DONNEES"
20 E$=INPUT$(1)
30 FOR I=1 TO 20
40 PRINT #2,I
50 NEXT I
60 E$=INPUT$(1)
70 CLOSE
```

Ce programme écrit 2Ø nombres sur la cassette. Le nombre d'octets écrits est inférieur à 128. Lancez l'exécution du programme et regardez les mouvements de la bande magnétique. (N'oubliez pas de mettre le magnétophone en enregistrement!...).

Un premier défilement est effectué pour écrire sur la bande un "bloc d'entête". Ce bloc mémorisera différentes informations: nom du fichier, programme BASIC, binaire ou ASCII, etc...

Appuyez sur une des touches du clavier. Rien ne se passe; les données ont été écrites dans la mémoire tampon. Celle-ci n'étant pas pleine, aucun transfert n'est effectué vers la bande magnétique.

Appuyez de nouveau sur une des touches du clavier, la bande magnétique se met en mouvement; les données contenues dans la mémoire tampon sont écrites sur la bande magnétique avec en plus des informations spéciales indiquant que le fichier est terminé.

Ce fichier pourra être relu avec le programme suivant. (Il faudra auparavant réembobiner votre bande et mettre le magnétophone en lecture).

```
10 OPEN "I",#10,"DONNEES"
20 FOR I=1 TO 20
30   INPUT #10,A
40   PRINT A;
50 NEXT I
60 CLOSE
```

Remarques:

1) Il n'est pas obligatoire d'avoir le même numéro de canal en lecture ou en écriture.

2) Pour la relecture la boucle 2Ø FOR I = 1 TO 2Ø peut être remplacé par l'utilisation de la fonction EOF (voir fonction EOF).

3) Avec la cassette vous ne pourrez travailler que sur un seul fichier à la fois.

PEEK: Fonction

PEEK ad

ad est une adresse mémoire.

La fonction PEEK renvoie le contenu de l'adresse mémoire ad. Sans l'extension mémoire 16 K la plus haute adresse mémoire est 32 767 soit 7FFF en hexadécimal. Avec l'extension 16 K cette adresse est 49 151 soit BFFF en hexadécimal.

Exemple: Le programme suivant imprime en hexadécimal le contenu d'une zone mémoire.

```
10 CLS
20 INPUT"AD DEBUT,AD FIN";A,B
30 FOR I=A TO B
40   PRINT HEX$(PEEK(I));" ";
50 NEXT I
```

Remarques:

1) Voir aussi chapitre ACCÈS AU LANGAGE MACHINE

2) L'adresse peut être négative, l'adresse choisie par le TO7 sera alors égale à 65 536 + adresse.

   Ainsi:

   PRINT PEEK (– 49) est équivalent à
   PRINT (PEEK (65 536 – 49)) = PRINT (PEEK (65 487))

PEN: Voir chapitre CRAYON OPTIQUE

PLAY: Instruction

PLAY c

c est une chaîne de caractère contenant la mélodie à jouer.

L'instruction PLAY permet de jouer de la musique. Vous disposez sur le TO7 de cinq octaves (O = 1 : la plus basse, 5 : la plus haute). Vous pouvez aussi faire varier la longueur de la note (L = 1: courte, 96 longue), le Tempo (T = 1 à 255) et l'attaque (A = Ø à 255).

Les notes sont inscrites "clairement" dans la chaîne de caractères: DO, RE, MI, FA, SO, LA, SI (seul SOL est tronqué en SO). Vous pouvez insérer dans votre mélodie des dièses (#), des bémols (b) et des silences P.

Les valeurs prises par défaut lors de la mise sous tension du TO7 sont:

Attaque Ø: (note continue)
Tempo: 5
Octave: 4
Longueur: 24 (noire)

Remarque : La chaîne de caractères ne doit comporter aucun espace. Les différentes notes peuvent être séparées par des points virgules.

Exemples :

1) 10 PLAY "DOREMIFASOLASIO5DOPD004SILASOFA
MIREDO"

2) 20 PLAY "DO;RE;MI;FA#;P;SOb;SO"

3) Le programme suivant vous permettra de tester l'effet des différents paramètres de l'instruction PLAY. A chaque exécution les paramètres sont remis à leur valeur implicite par le programme.

```
10 CLS
20 LOCATE 15,0 : PRINT "ESSAI DE PLAY"
30 LOCATE 0,5,0   INPUT "ENTRER VOTRE ME
LODIE ";M$
34 CONSOLE 7
36 CLS
40 PLAY "O4L24T5A0"
50 INPUT "ATTAQUE (0-255)";A
60 IF A<0 OR A>255 THEN 50
70 INPUT "OCTAVE (1-5)";O
80 IF O<1 OR O>5 THEN 70
90 INPUT "TEMPO (1-255)";T
100 IF T<1 OR T>255 THEN 90
110 INPUT "LONGUEUR (1-96)";L
120 IF L<1 OR L>255 THEN 110
130 A$=STR$(A)
140 A$=MID$(A$,2,3)
150 O$=STR$(O)
160 O$=MID$(O$,2,1)
170 L$=STR$(L)
180 L$=MID$(L$,2,2)
190 T$=STR$(T)
200 T$=MID$(T$,2,3)
210 PLAY "T"+T$+"A"+A$+"O"+O$+"L"+L$
220 PLAY M$
230 INPUT "VOULEZ-VOUS CONTINUER (O/N)";
R$
240 IF R$="O" THEN 36 ELSE PLAY "O4L24T5
A0" : CONSOLE 0 : END
```

Remarque: L'instruction PLAY admet comme paramètre une chaîne de caractères. Cette chaîne ne peut contenir aucun blanc sinon une erreur est détectée. Dans l'exemple 3, il faut donc enlever cet espace. Ceci est réalisé avec la fonction MID$. Nous éliminons ainsi systématiquement le premier caractère de la chaîne STR$(x) qui est toujours un espace. Ceci aurait pu aussi être réalisé avec l'instruction RIGHT$ de la façon suivante.

```
80 A$=STR$(A)
90 A$=RIGHT$(A$,LEN(A$)-1)
```

POINT: Voir chapitre POSSIBILITÉS GRAPHIQUES

POKE: Instruction

POKE a, x

a est une adresse mémoire

x est un nombre compris entre Ø ET 255.

L'instruction POKE écrit la valeur numérique de x dans la cellule mémoire ayant l'adresse a. L'adresse a devra être comprise entre 6 1ØØ hexa et 7FFF hexa sans extension mémoire et entre 6 1ØØ hexa et BFFF hexa si vous disposez d'une extension 16 K RAM.

Un POKE à une adresse comprise entre Ø ET 3FFFH n'aura aucun effet (cet espace mémoire est en effet occupé par la mémoire morte).

Un POKE a une adresse comprise entre 6 ØØØ H et 6ØFFH pourra par contre avoir un effet désastreux ! En effet cet espace est utilisé par le TO7 pour mémoriser différentes informations.

La mémoire écran occupe les adresses 4ØØØ H à 5FFFH.

Exemple: Le programme suivant vous permettra de modifier la mémoire à une adresse voulue. Pour arrêter le programme entrer une valeur négative en réponse à la question "NOUVEAU CONTENU".

```
10 CLS:SCREEN 2,0,0
20 INPUT "ADRESSE DEBUT";A
30 PRINT A;"CONTENU ACTUEL ";HEX$(PEEK(A
));
40 INPUT " NOUVEAU CONTENU";X
50 IF X<0 THEN END
60 IF X>255 THEN COLOR 1:PRINT"X DOIT ET
RE COMPRIS ENTRE 0 ET 255":COLOR 2:GOTO
40
70 POKE A,X   A=A+1
80 GOTO 30
```

Écrivez dans la mémoire écran avec ce programme!

Remarque: Comme pour PEEK, l'adresse peut être négative

POS: Fonction

POS

La fonction POS renvoie le numéro de la colonne (Ø à 39 ou Ø à 19 si on est en double largeur) dans laquelle se trouve le curseur. Cette fonction sera surtout utile pour la réalisation de programmes de saisie sur écran.

Exemple: Le programme suivant teste si le curseur se trouve à un endroit voulu de l'écran.

```
10 CLS
20 INPUT "POSITION CURSEUR (X,Y   X ENTR
E 15 ET 25 , Y EN 12)";X,Y
30 LOCATE X,Y,1
40 IF POS > 25 OR POS < 15 THEN PRINT "X
 INCORRECT "  ELSE IF CSRLIN=12 THEN PRI
NT "CORRECT" ELSE PRINT "Y INCORRECT"
```

Remarque: Voir aussi la fonction CSRLIN.

PRINT: Instruction

PRINT x_1 {;} x_2 {;} x_3 {;}...

x_1, x_2, x_3 ... sont des noms de variables, des constantes ou des expressions arithmétiques.

L'instruction PRINT permet l'affichage de vos résultats sur l'écran. La virgule provoque une tabulation toutes les 13 colonnes, alors que le point virgule donne un affichage contigu.

Les nombres positifs seront toujours précédés d'un espace lorsqu'ils seront affichés (signe + "transformé" en espace pour l'affichage).

Tous les nombres sont suivis de l'impression d'un espace.

Exemple:

```
10 CLS
20 J=1:A=2000
30 PRINT J;"JANVIER";A
```

Remarque 1: Si une chaine de caractères ne peut pas être affichée totalement sur la ligne courante, elle sera alors affichée sur la ligne suivante.

Exemple:

```
10 CLS
20 LOCATE 30,0: PRINT 12345;"CHAINE DE C
ARACTERES"
```

Le texte "CHAINE DE CARACTÈRES" est affiché sur la ligne en-dessous de 12 345 malgré le ";" qui signifiait: afficher juste après le nombre 12 345.

Remarque 2: Chaque fois que le BASIC du TO7 rencontre une virgule dans une instruction PRINT, il fait progresser le curseur jusque la prochaine tabulation. Ainsi l'espace entre le dernier caractère affiché et la nouvelle position du curseur sera remis à blanc. Toutefois ceci ne sera pas réalisé si l'affichage a lieu en fin de ligne (entre les colonnes 26 et 39).

Exemple: Dans cet exemple nous remplirons trois lignes consécutives avec le caractère "A" puis nous imprimerons quelque chose sur ces lignes.

```
1Ø CLS : SCREEN 2,Ø,Ø
2Ø LOCATE Ø,5,Ø
3Ø FOR I = 1 TO 12Ø
4Ø PRINT "A" ;
5Ø NEXT I
6Ø LOCATE Ø,5,Ø : COLOR 1,3
7Ø PRINT 1,
8Ø PRINT 2,,
9Ø PRINT 3,,,
```

Pour les lignes 1 et 2 tous se passe comme prévu par contre pour la ligne 3, la fin de la ligne n'est pas effacée.

Remarque: Quand vous travaillez en mode page, l'instruction PRINT efface la fin de la ligne courante après affichage.

Exemple:

```
1Ø CLS : SCREEN 2,Ø,Ø
2Ø LOCATE Ø,6,Ø : COLOR 1,3
3Ø PRINT "LIGNE 6"
4Ø CONSOLE ,,,2
5Ø LOCATE Ø,1Ø,Ø
6Ø PRINT "LIGNE 1Ø"
7Ø CONSOLE ,,,Ø
8Ø COLOR 2,Ø
```

Pour éviter cela il faut rajouter un point virgule dans l'instruction PRINT.

```
6Ø PRINT "LIGNE 1Ø" ;
```

— PRINT TAB (x)

x numéro de colonne.

La fonction TAB permet de spécifier une colonne particulière lors d'une impression. Il n'est pas possible de revenir en arrière sur une même ligne avec la fonction TAB.

Exemple:

```
PRINT "TAB" ; TAB(2Ø) ; "FONCTION"
```

Remarques :

1) La fonction TAB n'annule pas les effets des séparateurs , et ;

Ainsi :

```
1Ø PRINT "AAA" ; TAB(1Ø) ; "BBB"
```

donnera un résultat différent de 1Ø PRINT "AAA", TAB(1Ø), "BBB".

2) TAB(n) remet à blanc les colonnes Ø à n−1

Exemple :

```
1Ø CLS :
2Ø ? "AAA ... (20 fois) ..."
3Ø LOCATE Ø,Ø : COLOR ,, 1
4Ø PRINT TAB(1Ø) ; "x" ;
5Ø COLOR ,, 1
```

Avec ce petit programme nous nous apercevons que les 1Ø premières colonnes sont remises à blanc.

PRINT # nc, x_1 {;/,} , x_2 {;/,} ... .

nc est un numéro de canal.

x_1, x_2 variables.

L'instruction PRINT # permet d'écrire des données sur un fichier. Son fonctionnement est analogue à PRINT. Le fichier devra avoir été ouvert auparavant (voir instruction OPEN).

PSET : Voir chapitre POSSIBILITÉ GRAPHIQUE.

PTRIG : Voir utilisation du crayon optique.

READ : Instruction

READ x_1, x_2, x_3

x_1, x_2, x_3 : noms de variables.

L'instruction READ affecte aux variables x_1, x_2, x_3,... les valeurs spécifiées dans les instructions DATA.

Exemple :

```
1Ø CLS
2Ø READ A$, B
3Ø FOR I = 1 TO B
4Ø PRINT A$
5Ø NEXT I
6Ø DATA "TØ7 DE THOMSON", 1Ø
```

REM : Instruction

REM

L'instruction REM permet d'introduire un commentaire dans un programme. Tout ce qui suit une instruction REM est considéré comme commentaire même le séparateur ":".

Cette instruction peut être abrégée par une apostrophe '.

Exemple :

```
1Ø REM LES INSTRUCTIONS SUIVANTES SONT CONSIDEREES
   COMME COMMENTAIRE : PRINT "BONJOUR" : INPUT A

1ØØ X = SQR (A) : REM calcul de la racine carrée
2ØØ Y = V ↑ 3 : calcul de "V CUBE"
```

RESTORE

RESTORE n

n est un numéro de ligne. n peut être omis.

L'instruction RESTORE replace le pointeur de lecture de l'instruction READ au début de l'instruction DATA ayant le numéro de ligne n ou au début de la première instruction DATA du programme si n est omis.

Exemple:

```
10  CLS
20  READ A,B
30  PRINT "PREMIERE LECTURE ";A;B
40  RESTORE 120
50  READ X,F$,G
60  PRINT "SECONDE LECTURE ";X;F$;G
70  RESTORE
80  READ T,V,W,Z
90  PRINT "TROISIEME LECTURE ";T;V;W;Z
100 DATA 10,15
110 DATA 5
120 DATA 8,"TO7 DE THOMSON",1983
```

RESUME

RESUME { n / NEXT

n est un numéro de ligne et peut être omis.

L'instruction RESUME termine une séquence de traitement de l'erreur et continue l'exécution.

— à la première instruction de la ligne de programme n.
— à l'instruction suivant celle qui a provoqué l'erreur si NEXT est employé.
— à l'instruction qui a provoqué l'erreur si n est omis.

Remarque: Voir aussi ON ERROR GOTO, ERROR, ERR, ERL.

RETURN: Instruction

RETURN

L'instruction RETURN provoque le retour au programme appelant. C'est la dernière instruction exécutée dans un sous-programme.

Exemple: Voir instruction GOSUB.

RIGHT$: Fonction

RIGHT$ (c, n)

c est une chaîne de caractères
n est un nombre compris entre Ø et 255.

La fonction RIGHT$ renvoie les n derniers caractères de la chaîne de caractères c. Si n est nul la chaîne renvoyée sera une chaîne vide.

Exemple:

```
10 CLS:SCREEN 2,0,0
20 INPUT"CHAINE DE CARACTERES (A$)";A$
30 INPUT"NOMBRE DE CARACTERES ";N
40 IF N<0 OR N>255 THEN PRINT "LE NOMBRE
 DE CARACTERES DOIT ETRE COMPRIS ENTRE 0
 ET 255":GOTO 30
50 LOCATE 0,10,0 : PRINT "LA FONCTION RI
GHT$(A$,";N;") DONNE LA";" CHAINE DE CAR
ACTERES SUIVANTE :"; : COLOR 1 :PRINT RI
GHT$(A$,N):COLOR 2
60 INPUT "VOULEZ-VOUS CONTINUER (O/N)";R
$
70 IF R$="N" THEN END
80 INPUT "NOUVELLE CHAINE (O/N)";R$
90 CLS : IF R$="O" THEN GOTO 20 ELSE GOT
O30
```

RND: Fonction

RND

La fonction RND fournit un nombre aléatoire comprise entre Ø et 1, la valeur 1 ne pouvant jamais être atteinte.

RUN: Voir commande RUN.

SAVE: Voir commande SAVE.

SAVEM: Voir commande SAVEM.

SCREEN: Instruction

Voir chapitre POSSIBILITÉS GRAPHIQUES

SCREEN: Fonction

SCREEN (c, l)

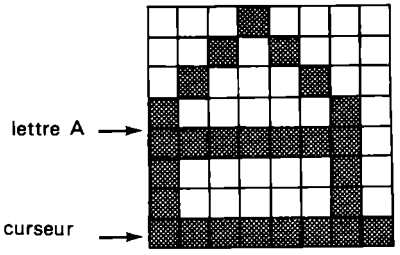
c numéro de colonne (Ø à 39)
l numéro de ligne (Ø à 24)

La fonction SCREEN renvoie le code ASCII du caractère situé aux coordonnées colonne, ligne spécifiées.

Si le numéro de colonne est supérieur à 39 il sera alors forcé à 39. De même le numéro de ligne sera forcé à 24 si il est supérieur à 24.

Les caractères graphiques utilisateur ne sont pas reconnus par la fonction SCREEN. La valeur retournée par SCREEN sera alors nulle. Il en sera de même chaque fois que le caractère scruté ne sera pas identifié.

Remarque importante: Éteignez le curseur avant d'utiliser la fonction SCREEN. En effet pour faire clignoter le curseur le TO7 affiche sur la dernière ligne du caractère une ligne de huit points consécutifs puis l'efface quelques instants plus tard:



Si lorsque le TO7 vient lire l'écran le curseur est allumé (voir dessin ci-dessus) le caractère ne sera alors pas reconnu et le code renvoyé sera ØØ. Il faut donc éteindre le curseur par l'instruction LOCATE Ø,Ø,Ø avant d'utiliser la fonction SCREEN.

Exemple: Dans ce premier exemple nous n'"éteignons" pas le curseur: appuyez sur une touche pour déclencher la lecture sur l'écran. De temps en temps le code renvoyé sera zéro (au lieu de 65).

```
10 CLS
20 PRINT"A"
30 LOCATE 0,0
50 E$=INPUT$(1)
60 X=SCREEN(0,0)    LOCATE 0,10    PRINT X
70 GOTO 30
```

Modifiez l'instruction 3Ø comme suit: LOCATE Ø,Ø,Ø et relancez l'exécution du programme. Le code renvoyé sera toujours 65.

Le programme suivant vous donnera les codes des lettres accentuées et du ç cédille.

```
10 CLS
20 A$="çàäâèéêëîïôöûüù"
30 LOCATE 0,0,0 'ETEINT CURSEUR
40 PRINT A$
50 J=-1
60 FOR I=1 TO LEN (A$) STEP 3
70 J=J+1
90 PRINT "LE CODE DE ";MID$(A$,I,3);" ES
T:";
100 COLOR 1 : PRINT SCREEN (J,0)
120 COLOR 2
130 NEXT I
```

Remarque: Dans ce programme nous avons pris deux compteurs I et J. L'indice de boucle I progresse de trois en trois. Souvenez-vous en effet, qu'une lettre accentuée occupe trois octets.

1 octet pour le code accent (ACC)
1 octet pour l'accent lui-même
1 octet pour la lettre (a, e, i, o, u).

De même pour pouvoir afficher ç ou à nous prenons trois octets consécutifs avec la fonction MID$.

SGN: Fonction

SGN (x)
x est une variable numérique.

La fonction SGN (signe) donne:

— 1 si x est négatif
Ø si x est nul
+ 1 si x est positif.

Exemple:

```
10 CLS
20 X=-20 : Y=0 : Z=+55
30 PRINT X ," SGN(";X;")=";SGN(X)
40 PRINT Y ," SGN(";Y;")=";SGN(Y)
50 PRINT Z ," SGN(";Z;")=";SGN(Z)
```

SIN: Fonction

SIN (a)

a est un angle exprimé en radians.

La fonction SIN calcul le sinus d'un angle exprimé en radians.

Exemple:

```
10 CLS
20 INPUT "DONNER VOTRE ANGLE EN DEGRES "
;A
30 TETA=3.14159*A/180    ' Conversion en
radians
40 PRINT " LE SINUS D'UN ANGLE DE";A;"DE
GRES EST";SIN(TETA)
```

SKIPF: Voir commande SKIPF

SPC

SPC (n)

n est une variable numérique.

La fonction SPC sera utilisée pour la présentation de résultats; elle permet en effet de laisser n espaces lors d'un affichage ou d'une impression.

Exemple: Le programme ci-dessous trace une diagonale sur l'écran.

```
10 CLS
20 PRINT "*"
30 FOR I=1 TO 15
40  PRINT SPC(I);"*"
50 NEXT
```

SQR: Fonction

SQR (x)

x est une variable numérique.

La fonction SQR renvoie la racine carrée de x.

```
10 CLS
20 INPUT "NOMBRE";N
30 IF N<0 THEN PRINT"ENTRER UN NOMBRE PO
SITIF" : GOTO 20
40 PRINT "LA RACINE CARREE DE";N;"EST :"
;SQR(N)
```

STICK : Voir chapitre EXTENSIONS

STOP : Instruction

STOP

L'instruction STOP termine l'exécution d'un programme. *Elle ne referme pas les fichiers.*

Pour reprendre l'exécution d'un programme arrêtée par STOP, il faudra utiliser la commande CONT au lieu de RUN. En effet la commande RUN referme les fichiers et remet toutes les variables du programme à zéro avant de commencer l'exécution du programme.

STR$: Fonction.

STR$ (x)
x est une variable numérique.

La fonction STR$ convertit la variable x en une chaîne alphanumérique.

Cette fonction sera très utilisée pour passer des paramètres à des fonctions ou des instructions utilisant des chaînes de caractères.

Exemples :

1)
```
10 CLS
20 INPUT "NOMBRE";N
30 N$=STR$(N)
40 PRINT N,N$
```

2) Voir exemple de *l'instruction PLAY.*

STRIG: Voir chapitre EXTENSIONS.

TAN: Fonction

TAN (x)

x est un angle exprimé en radians.

La fonction TAN calcule la tangente d'un angle exprimé en radians.

TRON
TROFF } Instructions

Les instructions TRON et TROFF permettent respectivement d'activer ou d'inhiber le mode TRACE. Dans ce mode le BASIC TO7 imprimera tous les numéros des lignes programme exécutées.

Ces instructions seront surtout utilisées lors de la mise au point d'un programme.

UNMASK: Voir chapitre POSSIBILITÉS GRAPHIQUES

USR: Voir chapitre ACCÈS AU LANGAGE MACHINE

VAL: Fonction

VAL (c)

c est une chaîne de caractère.

La fonction VAL convertit une chaîne de caractères en variable numérique. La conversion sera effectuée jusqu'au *premier caractère non numérique* de la chaîne c et en commençant par la gauche.

Exemple:

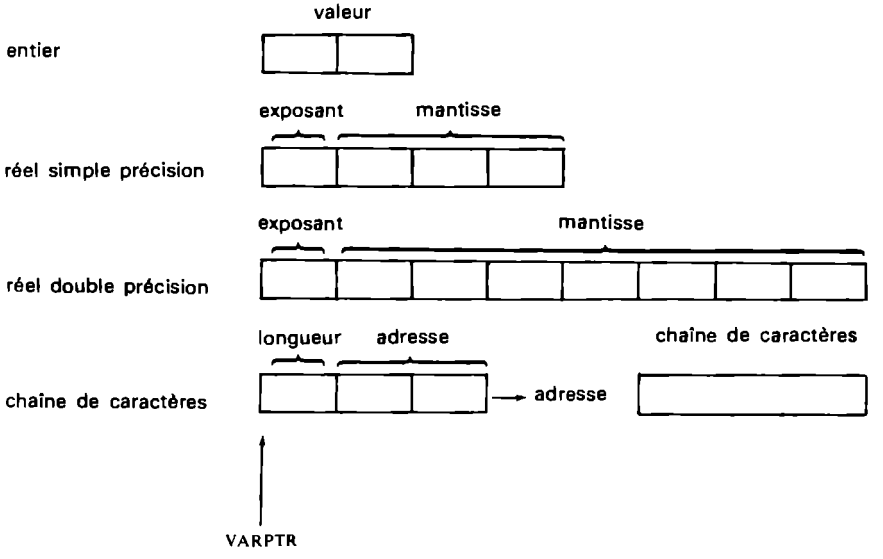
```
10 CLS
20 DATE$="01/05/2000"
30 JOUR=VAL(DATE$)
40 PRINT DATE$,JOUR
```

VARPTR : Fonction

VARPTR (x)

x est un nom de variable.

La fonction VARPTR pointe sur le premier octet contenant la valeur de x si x est un nom de variable numérique et sur le premier octet du descripteur de la chaîne de caractères x si x est un nom de chaîne de caractères. Ce descripteur d'une longueur de 3 octets, contient la longueur de la chaîne et l'adresse à laquelle elle est mémorisée.



Exemples :

1)
```
10 CLS
20 I%=347
30 A=VARPTR(I%)
40 X=256*PEEK(A)+PEEK(A+1)
50 PRINT "LA VARIABLE I% EST STOCKE A L'
ADRESSE";A;"SA VALEUR EST :";X
```

2)
```
10 CLS : SCREEN 2,0,0
20 A$="ESSAI DU BASIC TO7"
30 L=PEEK(VARPTR(A$)) ' L:LONGUEUR DE LA
 CHAINE A$
40 A=256*PEEK(VARPTR(A$)+1)+PEEK(VARPTR(
A$)+2)
45 PRINT "LA CHAINE DE CARACTERES EST LA
 SUIVANTE" : COLOR 1
50 FOR I=A TO A+L-1
60   PRINT CHR$(PEEK(I));
70 NEXT I
80 COLOR 2
90 PRINT "ELLE EST STOCKEE A L'ADRESSE";
A
```

2

Utilisation du crayon optique

Dans ce chapitre nous nous attacherons plus particulièrement aux instructions liées au crayon optique. Par contre, nous ne nous attarderons pas sur son principe de fonctionnement. Ceux qui désirent en savoir plus sur cette partie pourront se reporter au livre "Micro-ordinateurs comment ça marche?" paru dans la même collection.

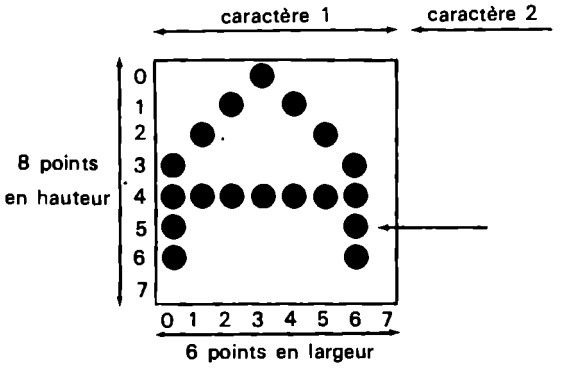
2.1. GÉNÉRALITÉS

2.1.1. Réglage du crayon optique

Avant toute utilisation du crayon optique il faudra, pour obtenir un fonctionnement correct, régler la luminosité de votre téléviseur. Pour cela revenons à la page de présentation en appuyant sur la touche "initialisation programme" ou en arrêtant et en remettant en marche votre TO7. Choisissons à présent le réglage du crayon optique en pressant la touche 3 au clavier (ou 2 si vous n'avez pas de K7). Nous pouvons alors procéder au réglage du crayon optique. Si vous avez quelques difficultés pour revenir à la page de présentation, il faut augmenter la luminosité de votre téléviseur et repointer le crayon optique au centre du petit carré blanc. Dès que celle-ci sera suffisante la page de présentation sera réaffichée. Vous pourrez alors faire votre choix en pointant le crayon optique dans le carré situé en face du numéro correspondant.

2.1.2. Précision du crayon optique

L'écran de votre téléviseur peut contenir 25 lignes de 40 caractères. Chaque caractère occupe huit points en largeur et huit points en hauteur.



Chacun de ces points sera reperé par sa place par rapport à une origine située en haut à gauche de l'écran. Ainsi si le caractère A dessiné ci-dessus est le premier caractère de l'écran le point repéré par la flèche aura les coordonnées suivantes : coordonnée horizontale 6, coordonnée verticale 5 (la première coordonnée a le numéro zéro. On donnera toujours la coordonnée horizontale en premier). Si le caractère est le dernier de l'écran, les coordonnées du point repéré seront (318, 198).

Le crayon optique nous permettra de pointer un point de l'écran et de récupérer les coordonnées du point ainsi choisi. Toutefois, avec cette méthode nous ne pouvons pas sélectionner tous les points de l'écran. En effet pour la coordonnée horizontale, le BASIC TO7 considérera toujours que le point choisi avec le crayon optique est le point situé au milieu du caractère correspondant.

Reprenons le caractère A dessiné précédemment en supposant que nous l'avons affiché en haut et à gauche de l'écran. Si nous réussissions à pointer le point repéré par la flèche, la coordonnée horizontale renvoyée aurait été 3. Si le caractère avait été affiché en bas et à droite de l'écran, la coordonnée horizontale aurait été 315.

Le programme suivant dessine deux diagonales avec des carrés blancs. Ces carrés ont la taille d'un caractère. Pointez chacun d'entre eux avec le crayon optique. La coordonnée horizontale sera affichée dans le bas de l'écran.

```
10 CLS:SCREEN 3,4,2
20 INPUT "AVEZ-VOUS REGLE LE CRAYON OPTI
QUE (O/N)";R$
30 IF R$="N" THEN PRINT "REGLEZ LE CRAYO
N OPTIQUE. ( RETOUR A LA PAGE MENU AVEC
LA TOUCHE 'INITIALISATION PROGRAMME')" E
LSE 50
40 E$=INPUT$(1):GOTO 40
50 CLS:COLOR ,7
60 FOR I=0 TO 19
70   LOCATE I,I,0:PRINT" "
80 NEXT
90 FOR I=20 TO 39
100  LOCATE I,I-20,0:PRINT" "
110 NEXT
120 INPUTPEN X,Y
130 COLOR 3,4:LOCATE 0,22,0:PRINT "LA CO
ORDONNEE HORIZONTALE EST :";X
140 PRINT "ARRET DU PROGRAMME PAR CNT C";
:GOTO 120
```

Pour la coordonnée horizontale, il n'y a aucun problème. La valeur renvoyée est le numéro de la ligne graphique (Ø à 199) sur laquelle on pointe.

Rajoutez au programme l'instruction suivante.

```
135 LOCATE0,23,0:PRINT "LA COORDONNEE VE
RTICALE EST :";Y
```

Dans cet exemple nous avons dessiné le tour de l'écran en vert. Cette zone est normalement inactive pour le crayon optique. Ainsi si vous appuyez le crayon optique dans la partie verte en haut ou en bas de l'écran la valeur renvoyée pour les coordonnées horizontale et verticale sera — 1. Par contre si vous appuyez le crayon optique sur le tour de l'écran à droite ou à gauche, les valeurs renvoyées seront :

— pour la coordonnée verticale : le numéro de la ligne,
— pour la coordonnée horizontale : 3 si vous avez pointé à gauche et 315 si vous avez pointé à droite.

Pour ce cas, il ne sera donc pas possible de savoir si l'utilisateur a pointé en dehors de la partie active de l'écran. Une solution consiste alors à ne pas utiliser la première et la dernière colonne de l'écran.

Remarque : Si le crayon optique est pointé en dehors de l'écran les valeurs renvoyées seront toujours — 1 (Appuyez le crayon optique sur votre table de travail par exemple). Il en sera de même si l'acquisition des coordonnées ne se fait pas correctement.

2.1.3. Couleur de zones pointées

Vous avez probablement remarqué que dans notre exemple, nous avons dessiné les cases en blanc. Ce n'est pas par hasard ! En effet toutes les couleurs ne conviennent pas avec le crayon optique et pour certaines d'entre elles, l'acquisition se révèle délicate. Modifiez l'instruction 5Ø comme suit.

```
50 INPUT"COULEUR (0-7)";C:CLS:COLOR,C
```

et essayez différentes couleurs en entrant un chiffre compris entre Ø et 7. Pour le noir et le rouge (C = Ø et C = 1), l'acquisition des coordonnées est délicate ! (La valeur renvoyée pour les coordonnées est toujours — 1).

Par contre pour les autres couleurs, aucune difficulté !

Remarque : Pour C = 4 les carrés sont dessinés en bleu, le fond de l'écran étant déjà bleu, vous ne les verrez pas apparaître ! Mais ils ont tout de même été dessinés !

2.1.4. Grandeur des zones de pointage

Après avoir essayé différentes cases, pointez la dernière case. En laissant la pointe du crayon optique appuyée contre le téléviseur et en changeant son orientation de droite à gauche vous obtenez des valeurs différentes pour la coordonnée horizontale. Il faudra donc, lorsque l'on désire pointer des zones avec le crayon optique, les choisir assez grandes. En effet si la taille des zones choisies est de un caractère, suivant la manière dont nous pointerons le crayon optique, nous pourrons obtenir la coordonnée du caractère suivant (ou du caractère précédent). Ce qui n'est pas toujours souhaitée !... De plus vous pourriez croire que le crayon optique de votre TO7 est en panne !

Ainsi il faudra prendre des zones ayant au moins deux caractères de large. (Une zone de deux caractères en largeur et en hauteur est parfaite !).

Ainsi pour une bonne utilisation du crayon optique il faudra :

- Régler la luminosité de votre téléviseur
- Se souvenir que la coordonnée horizontale renvoyée est celle du milieu du caractère pointée.
- Ne pas utiliser les couleurs noire et rouge.
- Définir des zones de pointage suffisamment grandes.

2.2. INSTRUCTIONS ET FONCTIONS RELATIVES AU CRAYON OPTIQUE

INPEN : Instruction

INPEN x, y

x, y sont des noms de variables numériques.

L'instruction INPEN renvoie dans la variable x la valeur de la coordonnée horizontale et dans la variable y la valeur de la coordonnée verticale d'un point de l'écran choisi avec le crayon optique. Avec cette instruction, le BASIC TO7 n'attend pas que l'interrupteur placé au bout du crayon optique soit fermé (crayon optique appuyé contre l'écran). Il faucra donc *systématiquement* s'assurer de la validité des coordonnées retournées en testant soit la valeur de x, soit la valeur de y.

Exemple: Avec le programme suivant testez les tremblements de votre main ! Laissez le crayon optique à environ quatre centimètres de l'écran et essayez de tracer une ligne verticale continue.

```
10 CLS:LOCATE0,0,0:SCREEN 4,6,6
20 INPEN X,Y
30 IF Y<0 THEN 20 ELSE PSET (X,Y)
40 GOTO 20
```

Ce programme peut être arrêté par [CNT] [C]. Le programme suivant vous permettra de réaliser quelques dessins. Son exécution sera arrêté par [CNT] [C].

```
10 CLS:LOCATE0,0,0:SCREEN 4,6,6:PSET(160,100)
20 INPEN X,Y
30 IF Y<0 THEN 20 ELSE LINE -(X,Y)
40 GOTO 20
```

INPUTPEN: Instruction

INPUTPEN x, y

x et y sont des noms de variables numériques.

L'instruction INPUTPEN a le même rôle que l'instruction INPEN. Toutefois avec INPUTPEN, l'acquisition des coordonnées ne sera faite que lorsque l'interrupteur situé au bout du crayon optique sera fermé (appuyé contre l'écran); l'exécution du programme sera donc suspendue tant que le BASIC TO7 ne détectera pas sa fermeture.

Exemples :

1) Reportez-vous à l'exemple donnée pour la fonction **INKEY$**. Nous allons réaliser la même chose avec **INPUTPEN**. Avouez que cela est beaucoup plus facile et tellement plus agréable d'utilisation !... En pointant sur les cases RAZ, POS, LEV et STOP vous pourrez effacer l'écran, poser ou lever le crayon. Le point de départ du dessin est au centre de l'écran.

```
10 CLS:SCREEN 3,4,4
20 LOCATE 0,1,0:PRINT " RAZ  POS  LEV  S
TOP "
30 BOX (3,4)-(35,19)
40 BOX (43,4)-(75,19)
50 BOX (83,4)-(115,19)
55 BOX (123,4)-(163,19)
60 PSET(160,100):C=1:L=0
70 CONSOLE 4
80 INPUTPEN X,Y
90 IF Y<4 THEN 80
100 IF Y<19 THEN IF 2<X AND X<36 THEN
CLS:C=0:GOTO 80 ELSE IF 42<X AND X<76 TH
EN L=0:C=0:GOTO 80 ELSE IF 82<X AND X<11
6 THEN L=1:GOTO 80 ELSE IF 122<X AND X<1
64 THEN 200 ELSE 80
110 IF L=1 THEN 120 ELSE IF C THEN LINE
-(X,Y) ELSE PSET (X,Y) : C=C+1
120 IF PTRIG THEN 120
130 GOTO 80
200 CONSOLE 0:LOCATE 0,4,0:SCREEN 4,6,6:
END
```

2) Le programme suivant vous permet de colorier les cases d'un rectangle avec différentes couleurs. Choisissez une couleur de la palette (à droite) en pointant le crayon optique dans la case située devant la couleur choisie. Cette case devient noire. Il ne vous reste plus qu'à pointer les cases du rectangle que vous désirez peindre avec cette couleur.

```
10 CLS:LOCATE39,20,0:SCREEN 4,6,6
15 C=0
20 FORI=0TO8
30 LINE(0,20*I)-(128,20*I),0
40 LINE(16*I,0)-(16*I,160),0
50 NEXTI
60 FORI=0TO7
70 BOX(240,20*I)-(255,20*I+20),0
80 BOXF(264,20*I+1)-(279,20*I+19),I
100 NEXTI
110 LOCATE0,21,0:PRINT"Pour arreter Poin
ter",,"dans la case blanche"
111 BOX(295,175)-(312,192),0
112 BOXF(296,176)-(311,191),7
115 INPUTPEN X,Y
117 IF X>294 AND X<313 AND Y>174 AND Y<1
93 THEN BEEP:END
120 IF X>239 AND X<256 AND Y<160 THEN BO
XF(240,20*C)-(255,20*C+20),-7:BOX(240,20
*C)-(255,20*C+20),0:C=INT(Y/20):BOXF(240
,20*C)-(255,20*C+19),0:BEEP
130 X=INT(X/16)
140 Y=INT(Y/20)
150 IF X<0 OR X>7 OR Y<0 OR Y>7 THEN 115
160 BOXF(16*X,20*Y)-(16*X+15,20*Y+19),C
165 BEEP
170 GOTO 115
300 BEEP
310 END
```

ONPEN GOTO
ONPEN GOSUB : Instructions

ONPEN GOTO nl_1, nl_2,...
ONPEN GOSUB nl_1, nl_2,...

nl_1, nl_2 sont des numéros de ligne du programme.

L'instruction ONPEN s'utilise toujours en coordination avec l'instruction PEN (un peu comme les instructions READ et DATA). L'instruction PEN permet de définir des zones "privilégiées" sur l'écran en attribuant à chacune d'entre elles un numéro (voir instruction PEN).

L'instruction ONPEN GOTO (ONPEN GOSUB) permet de se débrancher à la ligne programme dont le numéro est nl_1, nl_2,... (ou du sous-programme commençant à la ligne nl_1, nl_2,...) suivant que l'on pointe le crayon optique dans la zone privilégiée numéro Ø, numéro 1,.... Si l'on pointe le crayon optique dans une zone non privilégiée, l'instruction suivant l'instruction ONPEN sera exécutée.

Exemple: Voir instruction PEN.

Remarques:

1) L'instruction

```
1ØØ ONPEN GOTO 1ØØØ,2ØØØ est équivalente à

1ØØ INPUTPEN X,Y : IF X>=A1 AND X=<A2 AND Y
>=B1 AND Y=<B2 THEN GOTO 1ØØØELSE IF X>=A3
AND X=<A4 AND Y>=B3 AND Y<=B4 THEN
GOTO 2ØØØ
```

A1, A2, B1, B2 sont les coordonnées de la première zone.

A3, A4, B3, B4 sont les coordonnées de la seconde zone.

2) Attention avec ONPEN GOSUB.

En effet examinons la séquence suivante.

```
100 ONPEN GOSUB 1000,2000
110 'SUITE DU TRAITEMENT
```

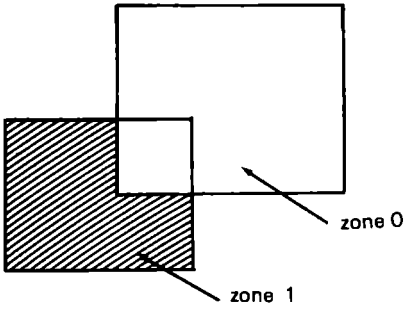
Si l'utilisateur pointe le crayon optique en dehors d'une zone "privilégiée", l'exécution du programme continuera avec l'instruction 11Ø et les instructions suivantes.

Si l'utilisateur pointe le crayon optique sur une zone privilégiée, le sous-programme placé en 1ØØØ ou en 2ØØØ sera exécuté mais lorsque un RETURN sera rencontrée, l'exécution se poursuivra avec l'instruction 11Ø. L'instruction 11Ø sera donc exécutée *dans tous les cas!*

3) L'instruction ONPEN réalise une acquisition des coordonnées x, y du point déterminé par le pointage du crayon optique. Toutefois ceci se passe de façon interne et il faudra une instruction INPUTPEN si vous désirez récupérer ces coordonnées.

4) Il est possible d'attribuer deux numéros différents à une même zone. Dans ce cas les débranchements seront toujours effectués au numéro de ligne qui correspond au plus petit numéro déclaré pour la zone. Il est de même lorsque deux zones se chevauchent.

Exemple:



Pour le BASIC TO7 la zone 1 est uniquement la zone hachurée ; le reste appartient à la zone Ø. Ceci peut être vérifié avec le petit programme suivant.

```
10 CLS:LOCATE 0,0,0
20 BOX(100,50)-(200,150)
30 BOX(25,100)-(125,175)
40 PEN0:(100,50)-(200,150),1:(25,100)-(1
25,175)
50 ONPEN GOTO 100,200
60 GOTO 50
100 PRINT"ZONE NUMERO ZERO"
110 END
200 PRINT"ZONE NUMERO UN"
210 END
```

5) Les zones imbriquées sont autorisées. Toutefois, la zone *la plus imbriquées* devra avoir *le plus petit numéro.* L'ordre de déclaration des zones dans le programme n'a aucune importance.

```
10 CLS
20 FOR I=0 TO 7
30 A=10*(I+1)
40 BOX(0,80-A)-(40*(I+1)-1,80+A)
50 PEN1:(0,80-A)-(40*(I+1)-1,80+A)
60 LOCATE I*5+2,10,0:PRINT STR$(I);
70 NEXT I
80 C=-1
90 ONPEN GOTO 170,160,150,140,130,120,
   110,100
```

```
95 GOTO 80
100 C=C+1
110 C=C+1
120 C=C+1
130 C=C+1
140 C=C+1
150 C=C+1
160 C=C+1
170 C=C+1
180 LOCATE0,22,0:PRINT "ZONE NUMERO";C
190 GOTO 80
```

6) Ne définissez pas une zone du type suivant PEN I; (C * 8 + 4, y_1) – (C * 8 + 10, y_2) ou c est une position caractère sur l'écran (c compris entre 0 et 39). En effet vous ne pourrez jamais pointer ces zones!... (Reportez-vous au paragraphe Généralités - Précision du crayon optique).

PEN: Instruction

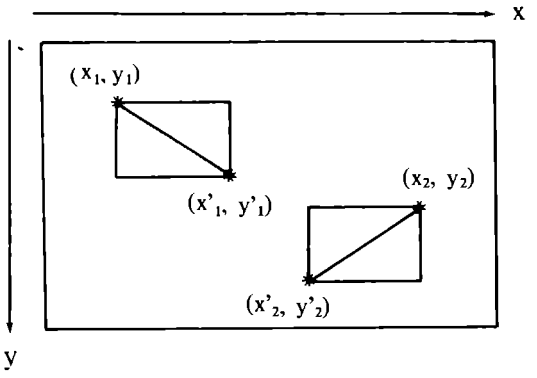
PEN n_1; (x_1, y_1) – (x_2, y_2), n_2; (x'_1, y'_1) – (x'_2, y'_2)

n_1 est un nombre compris entre 0 et 7

(x_1, y_1) et (x_2, y_2) déterminent deux points de l'écran (par leurs coordonnées).

L'instruction PEN permet de définir jusqu'à huit zones "privilégiées" sur l'écran. Chaque zone est reperée par un numéro (n_1, n_2,...).

Les zones définies sont rectangulaires. Les points (x_1, y_1), (x'_1, y'_1),... et (x_2, y_2), (x'_2, y'_2), ... sont les coordonnées des points diagonalement opposés des rectangles ainsi spécifiées (x_1, x'_1,)... sont donc compris entre 0 et 319 et (y_1, y'_1) entre 0 et 199.



Pour rebanaliser une zone de l'écran il suffira de supprimer les coordonnées des points repérant cette zone dans l'instruction PEN. La syntaxe de PEN devient alors PEN n_1 ;

Si l'instruction PEN n'est suivie d'aucuns numéros, toutes les zones privilégiées seront rebanalisées.

Exemple: Reprenons l'exemple donné pour l'instruction INPUTPEN mais cette fois ci au lieu de tester les coordonnées nous utiliserons l'instruction ONPEN.

```
10 CLS:SCREEN 3,4,4
20 LOCATE0,1,0:PRINT " RAZ  POS  LEV  ST
OP "
30 BOX (3,4)-(35,19)
40 BOX (43,4)-(75,19)
45 BOX (83,4)-(115,19)
50 BOX (123,4)-(163,19)
55 PEN0:(3,4)-(35,19)
60 PEN1:(43,4)-(75,19)
65 PEN2:(83,4)-(115,19)
70 PEN3:(123,4)-(163,19)
75 PEN4:(0,32)-(319,199)
80 PSET(160,100):C=1:L=0
85 CONSOLE 4
90 ONPEN GOTO 200,300,400,500,100
95 GOTO 90
```

```
100 INPUTPEN X,Y   IF Y<0 THEN 100
110 IF L=1 THEN 120 ELSE IF C THEN LINE
-(X,Y) ELSE PSET (X,Y) : C=C+1
120 IF PTRIG THEN 120
130 GOTO 90
200 CLS:C=0:GOTO 90
300 L=0:C=0:GOTO 90
400 L=1:GOTO 90
500 CONSOLE 0:LOCATE 0,0,0:SCREEN 4,6,6:
END
```

Remarques :

1) L'instructon PEN ne visualise pas les zones définies sur l'écran. Pour les repérer, utilisez l'instruction BOX (voir chapitre POSSIBILITÉS GRAPHIQUES).

2) Il est possible de spécifier des zones privilégiées de un caractère. La syntaxe de l'instruction PEN sera alors.

PEN n_1 ; (x, y), n_2 ; (x', y')

x, x',... et y, y',... seront alors respectivement les numéros de colonnes en caractère (compris entre Ø et 39) et les numéros de lignes (compris entre Ø ET 24). Toutefois nous vous déconseillons d'utiliser de telles zones. En effet leur pointage est délicat (voir paragraphe Généralités).

PTRIG : Fonction

PTRIG

La fonction PTRIG permet de savoir si l'interrupteur du crayon optique est fermé (appuyé contre l'écran) ou non. Si il est fermé, la valeur renvoyée est — 1.

Cette fonction est très utile lorsque plusieurs coordonnées doivent être acquises en suivant. Il sera alors possible de séparer l'acquisition de chaque point.

Exemples : Le programme suivant permet de tracer une ligne entre deux points choisis avec le crayon optique.

1)
```
10 CLS:SCREEN 4,6,6
20 INPUTPEN X,Y:IF X<0 THEN 20
30 PSET(X,Y)
40 INPUTPEN X,Y:IF X<0 THEN 40
50 LINE -(X,Y)
60 END
```

Lancez l'exécution de ce programme. Vous voyez OK apparaître avant d'avoir eu le temps de pointer l'autre extrémité de la ligne. En effet dès que l'acquisition du premier point a été faite, la suite du programme est exécutée c'est-à-dire l'acquisition du second point. Introduisons donc la fonction PTRIG pour nous permettre de retirer le crayon optique de l'écran.

```
35 IF PTRIG THEN 35
```

Ainsi nous bouclerons tant que le crayon optique ne sera pas décollé de l'écran.

2) La séquence de programme suivante permet d'attendre un appui touche ou un appui du crayon optique.

```
100 IF PTRIG THEN INPEN X, Y : IF x<0 THEN 100
    ELSE 110 ELSE A$:INKEY$ : IF A$= " "
    THEN 100 ELSE 110

110 REM Suite du programme
```

3

Accès au langage machine

La Programmation en langage machine consiste à programmer le microprocesseur 6 8Ø9 situé au cœur de votre TO7. Le but de ce chapitre n'est pas de vous apprendre le jeu d'instructions du microprocesseur 6 8Ø9 mais de vous montrer comment faire de la programmation en langage machine sur votre TO7. Ceux qui désirent apprendre et connaître le jeu d'instructions du microprocesseur 6 8Ø9 pourront se reporter à un prochain ouvrage de la collection qui traite ce sujet.

3.1. UTILISATION D'UN LANGAGE ASSEMBLEUR

Un langage ASSEMBLEUR, vendu sous forme de cartouche enfichable (encore appelée mémo 7), sera bientôt disponible pour le TO7.

Cet assembleur vous permettra de programmer directement le microprocesseur 6 8Ø9 en utilisant les mnémoniques des codes opérations. De plus y seront associées plusieurs fonctions facilitant la mise au point des programmes (points d'arrêts, examen de registres, ...)

3.2. CREATION D'UN PROGRAMME EN LANGAGE MACHINE

Pour créer un programme en langage machine il faut introduire, dans la mémoire du TO7, les codes binaires compréhensibles par le microprocesseur 6 8Ø9. Certaines instructions BASIC particulières permettront alors de venir exécuter ce programme en langage machine. Cependant lors de l'exécution d'un programme écrit en BASIC les variables utilisées dans le programme ne sont jamais stockées à une adresse fixe en mémoire; selon les opérations réalisées dans le programme, ces adresses sont modifiées.

Ainsi pour stocker un programme en langage machine il faut pouvoir disposer d'un endroit "sûr" (qui ne sera pas écrasé par les variables du programme écrit en BASIC). Deux méthodes sont possibles.

1) Utilisation d'une variable alphanumérique.

2) Réservation d'une place en haut de la mémoire (instruction CLEAR).

3.2.1. Utilisation d'une variable alphanumérique

On peut stocker le programme en langage machine dans une variable alphanumérique; chaque caractère de la chaîne sera alors un code binaire du programme écrit en langage machine. Cette variable sera alors gérée par le BASIC au même titre que les autres variables du programme. Pour retrouver l'adresse de cette variable en mémoire nous utiliserons la fonction BASIC VARPTR (voir chapitre 1). Toutefois, pour les chaînes de caractères, VARPTR ne renvoie pas directement l'adresse de la chaîne de caractères mais *l'adresse du descripteur de chaîne.* Ce descripteur contient la longueur de la chaîne et l'adresse mémoire du premier caractère. Reportez-vous à l'exemple 2 de la fonction VARPTR (chapitre 1). Les codes du programme en langage machine seront inclus dans une instruction DATA et pourront ainsi être lus par une instruction READ.

Exemple: Réalisons un programme en langage machine pour tracer une ligne horizontale et une ligne verticale au milieu de l'écran.

Pour cela nous travaillerons directement dans la mémoire écran. Celle-ci débute à l'adresse hexadécimale 4 ØØØ. Les quarante premiers octets de cette mémoire mémorisent l'état de la ligne graphique zéro, les quarante suivants, la ligne graphique un, etc... Nous aurons donc le programme suivant :

	LDX	# 4ØØØ H	8E	4Ø	ØØ
	PSHS	X	34	1Ø	
	LDA	# 1ØØ	86	64	
S	LEAX	4Ø, X	38	88	28
	DEC A		4A		
	BNE	S	26	FA	
	LDB	# 255	C6	FF	
	LDA	# 4Ø	86	28	
BOUCLE 1					
	STB	,X+	E7	8Ø	
	DECA		4A		
	BNE	BOUCLE 1	26	FB	
	PULS	X	35	1Ø	
	LDB	# 19	C6	13	
	ABX		3A		
	LDB	# Ø1H	C6	Ø1	
	LDA	# 2ØØ	86	C8	
BOUCLE 2					
	STB	, X	E7	84	
	LEAX	4Ø, X	3Ø	88	28
	DECA		4A		
	BNE	BOUCLE 2	26	F8	
	RTS		39		

Pour chaque ligne le point situé au milieu est le point dont la coordonnée horizontale est 159. Ce point correspond, dans la mémoire écran au dernier bit du dix neuvième caractère ($159 = 19 \times 8 + 7$). Nous nous plaçons donc sur le dix neuvième caractère (LDB #19, ABX) puis nous mettons à un le bit 7 de cet octet (STB, X avec B = Ø1). Chaque ligne ayant une longueur de 4Ø caractères, nous nous positionnerons au milieu de la ligne suivante en incrémentant le registre d'index X de 4Ø (LEAX 4Ø,X).

Le programme BASIC réalisant le chargement du programme écrit en langage machine sera alors le suivant.

```
10 REM     *** LANGAGE MACHINE ***
20 LM$="" ' INITIALISE LA CHAINE LM$
            A VIDE
30 FORI=1 TO 40
40 READ C
50 LM$=LM$+CHR$(C)
60 NEXTI
70 END


1000 DATA &H8E,&H40,&H00,&H34,&H10,&H86,
&H64,&H30,&H88,&H28,&H4A,&H26,&HFA,&HC6,
&HFF,&H86,&H28,&HE7,&H80,&H4A,&H26,&HFB,
&H35,&H10,&HC6,&H13,&H3A,&HC6,&H01,&H86,
&HC8,&HE7,&H84,&H30,&H88,&H28,&H4A,&H26,
&HF8,&H39
```

Lancez l'exécution de ce programme, la chaîne de caractères LM$ contient maintenant le programme écrit en langage machine. Vous pouvez le vérifier en affichant la chaine LM$.

```
2000 REM    **********************
2010 REM    *** VERIFICATION ***
2020 REM    **********************
2030 AD=256*(PEEK(VARPTR(LM$)+1))+PEEK(V
ARPTR(LM$)+2)
2040 FOR J=AD TO AD+39
2050 PRINT HEX$(PEEK(J));" ";
2060 NEXT J
2070 END
```

Remarques :

1) Avec cette méthode vous ne pourrez pas avoir de débranchements absolus. En effet la chaîne de caractères LM$ ne gardera pas une place fixe en mémoire. Il ne faudra donc utiliser que des débranchements relatifs.

2) Les chaînes de caractères ne peuvent pas excéder une longueur de 255 caractères. Le nombre d'octets de vos programmes écrits en langage machine sera donc limité à 255 octets au maximum.

3) N'utilisez pas l'instruction CLEAR après avoir chargé votre programme écrit en langage machine. En effet CLEAR réinitialise

toutes les variables du programme BASIC. Le programme en langage machine serait alors perdu.

3.2.2. Utilisation de l'instruction CLEAR

Lorsque vous mettez votre TO7 sous tension, celui-ci recherche la plus haute adresse mémoire utilisable. Cette adresse est 7FFFH (ou 32 767 en décimal) si vous ne disposez pas de l'extension mémoire 16K RAM et BFFFH avec l'extension 16 K RAM.

Avec l'instruction CLEAR on peut fixer la plus haute adresse mémoire utilisable par le BASIC, c'est le rôle du second paramètre de cette instruction.

Ainsi CLEAR, &H7FFØ fixera l'adresse 7FFØ (32 752 en décimal) comme la plus haute adresse utilisable par le BASIC. Les adresses 7FF1H à 7FFFH seront alors "ignorées" et ceci jusqu'à l'exécution d'une nouvelle instruction CLEAR ou la mise hors tension de votre TO7.

Ainsi la place mémoire réservée par une instruction CLEAR ne sera pas modifiée par le BASIC (*même par une instruction NEW!*).

Reprenons l'exemple précédent. Notre programme écrit en langage machine avait, une longueur de 4Ø octets. Nous fixerons donc la plus haute adresse utilisable par le BASIC à:

7FFFH – 28H = 7FD7H

(BFFFH – 28H = BFD7H si vous avez une extension 16 K RAM) et nous stockerons notre programme à partir de l'adresse hexadécimale 7FD8H (N'oubliez pas que l'adresse 7FD7H est utilisée par le BASIC ; revoyez l'instruction CLEAR au chapitre 1).

Notre programme de chargement sera alors.

```
10 CLEAR ,&H7FD7    ' RESERVE 40 OCTETS
  ( &H7FD8-&H7FFF ) POUR PROGRAMME ECRIT
  EN LANGAGE MACHINE
12 ' CLEAR &HBFD7       ' (AVEC 16K RAM)
20 FOR I=1 TO 40
30 READ C
40 POKE &H7FD7+I,C
45 ' POKE &HBFD7+I,C   ' (AVEC 16K RAM)
50 NEXT I
60 END
```

```
1000 DATA &H8E,&H40,&H00,&H34,&H10,&H86,
&H64,&H30,&H88,&H28,&H4A,&H26,&HFA,&HC6,
&HFF,&H86,&H28,&HE7,&H80,&H4A,&H26,&HFB,
&H35,&H10,&HC6,&H13,&H3A,&HC6,&H01,&H86,
&HC8,&HE7,&H84,&H30,&H88,&H28,&H4A,&H26,
&HF8,&H39
```

Remarque: Avec cette méthode nous pourrons utiliser des débranchements absolus dans le programme en langage machine mais dans ce cas le programme ne sera plus relogeable et son implantation dépendra de l'espace mémoire dont vous disposez (avec ou sans bloc 16 K RAM) alors que dans le cas précédent celle-ci était gérée par le BASIC.

Le programme de vérification devient alors:

```
2000 REM   **********************
2010 REM   *** VERIFICATION ***
2020 REM   **********************
2030 AD=&H7FD8 ' AD=&HBFD8 AVEC 16K RAM
2040 FOR J=AD TO AD+39
2050 PRINT HEX$(PEEK(J));" ";
2060 NEXT J
2070 END
```

3.3. QUELQUES PRÉCAUTIONS

— Pour revenir correctement au programme BASIC il faut obligatoirement terminer votre sous-programme écrit en langage machine par une instruction RTS (code hexadécimal 39).

— Le 6 8Ø9 possède un *registre de page DP*. Ce registre de page est utilisé par le moniteur du TO7. Il ne faudra donc pas le modifier. Si toutefois vous vous servez de ce registre il faudra le réinitialiser avant le retour au programme BASIC.

— De même (si vous utilisez) la *pile système* S, veillez à la réinitialiser correctement avant le retour au BASIC.

3.4. EXÉCUTION DU PROGRAMME ÉCRIT AU LANGAGE MACHINE

Deux instructions permettent de venir exécuter un programme écrit en langage machine EXEC et USR.

3.4.1. EXEC

EXEC: Instruction

EXEC ad

ad est l'adresse d'un sous-programme écrit en langage machine.

L'instruction EXEC permet de venir exécuter un programme écrit en langage machine, *sans passage de paramètres* entre le programme écrit en BASIC et le programme écrit en langage machine.

Exemple: Pour venir exécuter l'exemple précédent nous rajouterons à notre programme BASIC l'instruction.

```
55 EXEC ØH7FD8
```

Pour être sûr de bien rester dans la mémoire forme dans notre programme nous rajouterons avant l'appel du sous-programme écrit en assembleur l'instruction BASIC suivante.

```
52 CLS : POKE &HE7C3,1 OR PEEK (&HE7D3)
```

Remarque: Nous aurions pu inclure ceci dans notre programme écrit en langage machine. Faites le vous même pour vous entraîner un peu!

Le listage complet de notre programme est donc.

```
10 CLEAR ,&H7FD7    ' RESERVE 40 OCTETS
  ( &H7FD8-&H7FFF ) POUR PROGRAMME ECRIT
  EN LANGAGE MACHINE
12 ' CLEAR &HBFD7       ' (AVEC 16K RAM)
20 FOR I=1 TO 40
30 READ C
40 POKE &H7FD7+I,C
45 ' POKE &HBFD7+I,C   ' (AVEC 16K RAM)
50 NEXT I
52 CLS:POKE &HE7C3,1 OR PEEK(&HE7C3)
55 EXEC &H7FD8
60 END
1000 DATA &H8E,&H40,&H00,&H34,&H10,&H86,
&H64,&H30,&H88,&H28,&H4A,&H26,&HFA,&HC6,
&HFF,&H86,&H28,&HE7,&H80,&H4A,&H26,&HFB,
&H35,&H10,&HC6,&H13,&H3A,&HC6,&H01,&H86,
&HC8,&HE7,&H84,&H30,&H88,&H28,&H4A,&H26,
&HF8,&H39
2000 REM    **********************
2010 REM    *** VERIFICATION ***
2020 REM    **********************
2030 AD=&H7FD8 ' AD=&HBFD8 AVEC 16K RAM
2040 FOR J=AD TO AD+39
2050 PRINT HEX$(PEEK(J));" ";
2060 NEXT J
2070 END
```

3.4.2. Passage de paramètres avec EXEC

Si l'on désire passer des valeurs entre le programme écrit en BASIC et le programme écrit en langage machine il faudra alors réserver de la place supplémentaire en mémoire pour venir stocker la valeur des paramètres. Ces places réservées seront initialisées avant l'appel du sous-programme avec l'instruction POKE et le résultat sera récupéré avec la fonction PEEK.

Exemple: Écrivons en langage machine un programme très simple. Multiplier un nombre donné par deux; le nombre sera compris entre Ø et 127. Le programme sera.

	LDA	NOMBRE	B6	7F	FF
	ASLA		48		
	STA	NOMBRE	B7	7F	FF
	RTS		39		
NOMBRE	ØØ		ØØ		

Ce programme fait huit octets. Nous réserverons un octet supplémentaire pour mémoriser la valeur du nombre à multiplier par deux. Il faudra donc réserver neuf octets pour le programme en langage machine. Nous aurons donc le programme définitif suivant.

```
10 REM MULTIPLICATION PAR 2
20 DEFINT N
30 CLEAR ,&H7FFF-&H09
40 A=&H7FFF-&H09
50 FOR I=A+1 TO A+8
60 READ C
70 POKE I,C
80 NEXT I
90 CLS:INPUT"NOMBRE A MULTIPLIER ";N
100 POKE A+9,N
110 EXEC A+1
120 PRINT "LE RESULTAT EST";PEEK(A+9)
130 INPUT "ENCORE (O/N) ";R$
140 IF R$="O" THEN 80 ELSE END
500 DATA &HB6,&H7F,&HFF,&H48,&HB7,&H7F,
        &HFF,&H39,&H00
```

Remarque : Cette méthode ne vous permet pas de travailler aisément sur des nombres réels. Il faudrait en effet recopier les quatres octets dans lesquels ce nombre est mémorisé. Pour les nombres réels, il sera donc préférable de "passer" l'adresse à laquelle ces nombres sont rangés en mémoire. (Rappelez-vous celle-ci peut être obtenue en utilisant la fonction VARPTR).

Création du programme
un langage machine


```
100 ' Passage de l'adresse de la variabl
e reelle B
110 D=VARPTR(B)
120 X1=INT(D/256)
130 X2=D-256*INT(D/256)
140 POKE Y,X1
150 POKE Y+1,X2
```

y est l'adresse, dans votre programme en langage machine, à laquelle vous sauvegardez l'adresse du paramètre B.

3.5. DEFUSR ET USR

L'instruction USR permet de passer *directement* un paramètre entre le programme BASIC et le sous-programme écrit en langage machine.

DEFUSR

DEFUSR n = a

n est un numéro compris entre Ø et 9
a est une adresse mémoire.

L'instruction DEFUSR permet de spécifier l'adresse du premier octet d'un sous-programme écrit en langage machine (ie son point d'entrée).

USR

USR n, (x)

n est un numéro compris entre Ø et 9
x est un nom de variable ou une constante.

L'instruction USR permet de venir exécuter un sous-programme écrit en langage machine avec passage d'un paramètre. n permet d'indiquer le programme en langage machine choisi, en correspondance avec l'instruction DEFUSR.

Le passage des paramètres est effectué dans une zone spéciale située à l'adresse 6 155. L'adresse de cette zone est donnée par le registre X ; l'accumulateur A spécifie le type du paramètre (entier, réel, ...). Vérifions cela avec le programme suivant.

	STX	ADRESSE	BF	7F	F9	
	STA	TYPE	B7	7F	FB	
	LDY	# PARAM	1Ø	8E	7F	FC
STOCKE	LDA	#4	86	Ø4		
	LDB	, X +	E6	8Ø		
	STB	, Y +	E7	AØ		
	DECA		4A			
	BNE	STOCKE	26	F9		
	RTS		39			
ADRESSE	ØØ ØØ		ØØ	ØØ		
TYPE	ØØ		ØØ			
PARAM	ØØ ØØ ØØ ØØ		ØØ	ØØ	ØØ	ØØ

Remarque : Ce programme stocke aussi la valeur du paramètre. Ceci doit en effet être réalisé par le sous-programme écrit en langage machine car la zone où le BASIC range le paramètre pour le programme écrit en langage machine est temporaire. Cette zone sera utilisée pour d'autres opérations par le BASIC. Son contenu est donc détruit : vérifiez le de la façon suivante :

	STX	ADRESSE	BF	7F	FE
	STA	TYPE	B7	7F	FD
	RTS		39		
TYPE	Ø		ØØ		
ADRESSE	Ø		ØØ	ØØ	

Après appel de ce petit sous-programme, récupérez la valeur de l'adresse et du type avec la fonction PEEK puis lisez le contenu de cette zone.

Exemple:

```
5 CLS
10 DEFUSR5=&H7FF6
20 CLEAR ,&H7FF5
30 FOR A=&H7FF6 TO &H7FFF
40    READ C:POKE A,C
50 NEXT
60 X=USR(4%)
70 TYPE=PEEK(&H7FFD)
80 ADRESSE=256*PEEK(&H7FFE)+PEEK(&H7FFF)
90 PRINT "LE CONTENU DE ADREESE EST ";HE
X$(PEEK(ADRESSE));" ";HEX$(PEEK(ADRESSE+
1))
100 END
500 DATA &HBF,&H7F,&HFE,&HB7,&H7F,&HFD,
&H39,&H00,&H00,&H00
```

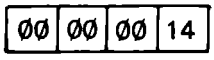
Vous n'obtenez pas 4

Revenons à notre programme complet. Nous nous apercevons que la valeur du type est en réalité le nombre d'octets de l'argument (2 pour les entiers, 4 pour les réels, 8 pour les nombres en double précision et 3 pour les chaînes de caractères — l'argument renvoyé étant alors le descripteur de la chaîne).

```
10 CLEAR ,&H7FFF-27
20 DEFUSR=&H7FFF-26
30 FOR A=&H7FFF-26 TO &H7FFF
40   READ C
50   POKE A,C
60 NEXT
65 A$="Langage machine"
70 CLS
80 X%=USR(20%)
90 GOSUB 1000
100 X=USR(20.0)
110 GOSUB 1000
120 X#=USR(20#)
130 GOSUB 1000
140 X$=USR(A$)
150 GOSUB 1000
160 END
```

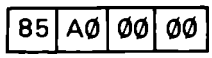
```
500 DATA &HBF,&H7F,&HF9,&HB7,&H7F,&HFB,
&H10,&H8E,&H7F,&HFC,&H86,&H04,&HE6,&H80,
&HE7,&HA0,&H4A,&H26,&HF9,&H39,&H00,&H00,
&H00,&H00,&H00,&H00,&H00
1000 PRINT "ADRESSE :";HEX$(256*PEEK(&H7
FF9)+PEEK(&H7FFA))
1010 PRINT "TYPE :";HEX$(PEEK(&H7FFB))
1020 PRINT "PARAMETRE :";HEX$(PEEK(&H7FF
C));" ";HEX$(PEEK(&H7FFD));" ";HEX$(PEEK
(&H7FFE));" ";HEX$(PEEK(&H7FFF))
1030 RETURN
```

Nous pouvons ainsi vérifier les valeurs passées. Dans le cas ou le paramètre est un entier, sa valeur est mémorisé dans les deux derniers octets de la zone tampon. Ainsi lorsque l'on travaillera avec des nombres entiers il faudra incrémenter de deux l'adresse contenue dans le registre X.



début de la zone tampon — nombre entier

Pour les nombres réels simple ou double précision, la valeur du paramètre est stockée sous forme virgule flottante mantisse et exposant.



exposant | mantisse

Nous retrouvons bien la valeur de l'argument donné. En effet 85 est l'exposant: la mantisse sera multipliée par $2^5 = 32$.

AØ est la mantisse: ce qui donne une valeur de Ø,625.

$$A\emptyset = 1 \times 2^{-1} + \emptyset \times 2^{-2} + 1 \times 2^{-3} = \emptyset.5 + \emptyset + \emptyset.125 = \emptyset,625$$

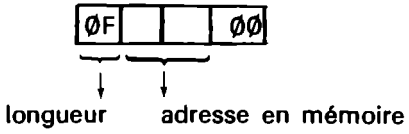
la valeur de l'argument est donc

$$32 \times \emptyset,625 = 2\emptyset.\emptyset$$

Cette représentation ne tient pas compte du signe. En effet, prenons $-2\emptyset$ comme argument: Nous obtiendrons la même valeur pour l'argu-

ment (85 AØ ØØ ØØ). En réalité le signe de l'argument est mémorisé à l'adresse hexadécimale 615D (24 925 en décimal) c'est-à-dire huit octets après l'exposant.

Pour les chaînes de caractères, le paramètre qui est passé est le descripteur de zones pointé par X. Celui-ci contient donc la longueur de la chaîne et l'adresse où elle est stockée en mémoire.



Dans notre exemple les arguments passés sont: longueur de la chaîne 15 octets (ØF), adresse de la chaîne 6656H. Vérifions-le avec la séquence suivante.

```
3000 FOR A=&H6656 TO &H6656+14
3010    PRINTCHR$(PEEK(A));
3020 NEXT A
```

Lancer son exécution avec la commande GOTO 3 ØØØ.

Remarque sur les chaînes de caractères

Si vous travaillez sur des chaînes de caractères dans un programme écrit en langage machine, n'augmentez pas sa longueur. En effet quand vous initialisez une chaîne, le BASIC TO7 force l'adresse du descripteur de zone avec l'adresse d'initialisation de cette variable chaîne dans la liste du programme.

Exemple: Vérifions ceci en BASIC. Transformons BONJOUR en BONSOIR.

```
10 CLS
20 A$="BONJOUR"
30 AD=256*PEEK(VARPTR(A$)+1)+PEEK(VARPTR
(A$)+2)
40 AD=AD+3
50 POKE AD,ASC("S")
60 POKE AD+2,ASC("I")
```

Si vous demandez un listage de votre programme vous obtenez alors le résultat suivant.

```
10 CLS
20 A$="BONSOIR"
30 AD=256*PEEK(VARPTR(A$)+1)+PEEK(VARPTR
(A$)+2)
40 AD=AD+3
50 POKE AD,ASC("S")
60 POKE AD+2,ASC("I")
```

BONJOUR est devenu BONSOIR dans le programme lui-même! Le même phénomène se produira avec un programme écrit en langage machine. Le programme suivant transformera BONJOUR en BONSOIR.

LEAX	[1,X]	3Ø	98	Ø1	(x) = adr chaîne de caractères
LEAX	3,X	3Ø	Ø3		(x) = adr du J
LDB	#53H	C6	53		(B) = code ASCII de lettre S
STB	,X+ +	E7	81		(x) pointe maintenant sur U
LDB	#49H	C6	49		(B) = code ASCII de lettre I
STB	,X	E7	84		
RTS		39			

LEAX [1,X] est un adressage indirect indexé. Nous utilisons un adressage indexé car l'adresse renvoyée dans x au moment de l'appel est l'adresse du descripteur de zones. Celui-ci fait trois octets. Le premier étant la longueur de la chaîne et les deux suivants l'adresse où elle est stockée en mémoire. Le 6 8Ø9 calculera donc d'abord l'adresse (X) + 1 puis prendra le contenu de cette adresse comme adresse effective. X pointera alors sur le premier caractère de la chaîne.

Le programme complet sera donc le suivant.

```
10 CLS:CLEAR ,&H7FF1
20 DEFUSR=&H7FF2
30 A$="BONJOUR"
40 FOR AD=&H7FF2 TO &H7FFF
50     READ C:POKE AD,C
60 NEXT
70 B$=USR(A$)
90 DATA &H30,&H98,&H01,&H30,&H03,&HC6,
&H53,&HE7,&H81,&HC6,&H49,&HE7,&H84,&H39
```

Pour éviter ce petit problème il faut donc faire recopier par le TO7 la variable chaîne dans la zone des variables. Il suffit donc d'insérer avant l'instruction 60 l'instruction.

```
55 A$ = A$ +""
```

Le BASIC TO7 reconnaîtra alors une concaténation des chaînes et recopiera cette variable dans la zone des variables. Toutefois il faudra veiller à ne pas dépasser la longueur de la chaîne de caractères, ou l'on risque de venir écraser d'autres variables.

Pour le retour au programme BASIC, le passage des paramètres s'effectue de la même manière. Il faudra donc initialiser le registre A avec le type du paramètre (2: entier, 3: chaîne de caractères, 4: réel simple précision, 8: réel double précision). Il n'est toutefois pas nécessaire d'initialiser le registre X avec l'adresse de la zone où sont rangés les paramètres. Cette adresse est supposée être 6155H par le BASIC.

Exemple: Le programme suivant renverra dans la variable A, l'opposé de la variable B.

```
10 SCREEN ,,0:COLOR 2,0:CLS
20 CLEAR ,&H7FFF-7
30 DEFUSR=&H7FFF-6
40 FOR A=&H7FFF-6 TO &H7FFF
50     READ C:POKE A,C
60 NEXT
70 INPUT " VALEUR DE B ";B
80 A=USR(B)
90 COLOR 3,4:PRINT "LA VALEUR DE A EST "
;A:COLOR 2,0
100 DATA &HE6,&H08,&HC8,&H80,&HE7,&H08,
&H39
```

Pour transformer un nombre réel en son opposé il suffit de complémenter son signe. Celui-ci est mémorisé à l'adresse 615DH. Lors de l'appel du sous-programme écrit en langage machine. Le BASIC range la valeur de la variable B dans la zone située à l'adresse 6155H et initialise les registres A et X avec les valeurs appropriées. Nous ne devrons donc pas les modifier.

LDB	8,X	E6	ØB
EORB	#8ØH	C8	8Ø
STB	8,X	E7	ØB
RTS		39	

ANNEXE

Code des instructions du 6 8Ø9

Code hexa	Mnemonique	Mode
ØØ	NEG	DIRECT
Ø3	COM	
Ø4	LSR	
Ø6	ROR	
Ø7	ASR	
Ø8	ASL/LSL	
Ø9	ROL	
ØA	DEC	
ØC	INC	
ØD	TST	
ØE	JMP	
ØF	CLR	DIRECT
12	NOP	INHERENT
13	SYNC	INHERENT
16	LBRA	RELATIF
17	LBSR	RELATIF
19	DAA	INHERENT
1A	ORCC	IMMEDIAT

Code hexa	Mnemonique	Mode
44	LSRA	INHERENT
46	RORA	
47	ASRA	
48	ASLA/LSLA	
49	ROLA	
4A	DECA	
4C	INCA	
4D	TSTA	
4F	CLRA	
5Ø	NEGB	
53	COMB	
54	LSRB	
56	RORB	
57	ASRB	
5B	ASLB/LSLB	
59	ROLB	
5A	DECB	
5C	INCB	INHERENT

Code hexa	Mnemonique	Mode
1C	ANDCC	IMMEDIAT
1D	SEX	INHERENT
1E	EXG	IMMEDIAT
1F	TFR	IMMEDIAT
2Ø	BRA	RELATIF
21	BRN	↕
22	BHI	↕
23	BLS	↕
24	BHS/BCC	↕
25	BLO/BCS	↕
26	BNE	↕
27	BEQ	↕
28	BVC	↕
29	BVS	↕
2A	BPL	↕
2B	BMI	↕
2C	BGE	↕
2D	BLT	RELATIF

Code hexa	Mnemonique	Mode
5D	TSTB	INHERENT
5F	CLRB	INHERENT
6Ø	NEG	INDEXE
63	COM	↕
64	LSR	↕
66	ROR	↕
67	ASR	↕
6B	ASL/LSL	↕
69	ROL	↕
6A	DEC	↕
6C	INC	↕
6D	TST	↕
6E	JMP	↕
6F	CLR	INDEXE
7Ø	NEG	ETENDU
73	COM	↕
74	LSR	↕
76	ROR	ETENDU

Code hexa	Mnemonique	Mode
2E	BGT	RELATIF
2F	BLE	RELATIF
3Ø	LEAX	INDEXE
31	LEAY	↕
32	LEAS	↕
33	LEAU	INDEXE
34	PSHS	IMMEDIAT
35	PULS	↕
36	PSHU	↕
37	PULU	IMMEDIAT
39	RTS	INHERENT
3A	ABX	↕
3B	RTI	INHERENT
3C	CWAI	IMMEDIAT
3D	MUL	INHERENT
3F	SWI	↕
4Ø	NEGA	↕
43	COMA	INHERENT

Code hexa	Mnemonique	Mode
77	ASR	ETENDU
7B	ASL/LSL	↕
79	ROL	↕
7A	DEC	↕
7C	INC	↕
7D	TST	↕
7E	JMP	↕
7F	CLR	ETENDU
8Ø	SUBA	IMMEDIAT
81	CMPA	↕
82	SBCA	↕
83	SUBD	↕
B4	ANDA	↕
85	BITA	↕
86	LDA	↕
88	EORA	↕
89	ADCA	↕
8A	ORA	IMMEDIAT

Code hexa	Mnemonique	Mode
8B	ADDA	IMMEDIAT
BC	CMPX	IMMEDIAT
BD	BSR	RELATIF
BE	LDX	IMMEDIAT
9Ø	SUBA	DIRECT
91	CMPA	↑
92	SBCA	|
93	SUBD	|
94	ANDA	|
95	BITA	|
96	LDA	|
97	STA	|
9B	EORA	|
99	ADCA	|
9A	ORA	|
9B	ADDA	|
9C	CMPX	|
9D	JSR	DIRECT

Code hexa	Mnemonique	Mode
C2	SBCB	IMMEDIAT
C3	ADDD	↑
C4	ANDB	|
C5	BITB	|
C6	LDB	|
CB	EORB	|
C9	ADCB	|
CA	ORB	|
CB	ADDB	|
CC	LDD	|
CE	LDU	IMMEDIAT
DØ	SUBB	DIRECT
D1	CMPB	↑
D2	SBCB	|
D3	ADDD	|
D4	ANDB	|
D5	BITB	|
D6	LDB	DIRECT

Code hexa	Mnemonique	Mode
9E	LDX	DIRECT
9F	STX	DIRECT
AØ	SUBA	INDEXE
A1	CMPA	↑
A2	SBCA	|
A3	SUBD	|
A4	ANDA	|
A5	BITA	|
A6	LDA	|
A7	STA	|
AB	EORA	|
A9	ADCA	|
AA	ORA	|
AB	ADDA	|
AC	CMPX	|
AD	JSR	|
AE	LDX	|
AF	STX	INDEXE

Code hexa	Mnemonique	Mode
D7	STB	DIRECT
DB	EORB	↑
D9	ADCB	|
DA	ORB	|
DB	ADDB	|
DC	LDD	|
DD	STD	|
DE	LDU	|
DF	STU	DIRECT
EØ	SUBB	INDEXE
E1	CMPB	↑
E2	SBCB	|
E3	ADDD	|
E4	ANDB	|
E5	BITB	|
E6	LDB	|
E7	STB	|
EB	EORB	INDEXE

Code hexa	Mnemonique	Mode
BØ	SUBA	ETENDU
B1	CMPA	
B2	SBCA	
B3	SUBD	
B4	ANDA	
B5	BITA	
B6	LDA	
B7	STA	
BB	EORA	
B9	ADCA	
BA	ORA	
BB	ADDA	
BC	CMPX	
BD	JSR	
BE	LDX	
BF	STX	ETENDU
CØ	SUBB	IMMEDIAT
C1	CMPB	IMMEDIAT

Code hexa	Mnemonique	Mode
E9	ADCB	INDEXE
EA	ORB	
EB	ADDB	
EC	LDD	
ED	STD	
EE	LDU	
EF	STU	INDEXE
FØ	SUBB	ETENDU
F1	CMPB	
F2	SBCB	
F3	ADDD	
F4	ANDB	
F5	BITB	
F6	LDB	
F7	STB	
F8	EORB	
F9	ADCB	
FA	ORB	ETENDU

Code hexa	Mnemonique	Mode
FB	ADDB	ETENDU
FC	LDD	
FD	STD	
FE	LDU	
FF	STU	ETENDU
1Ø21	LBRN	RELATIF
1Ø22	LBHI	
1Ø23	LBLS	
1Ø24	LBHS/LBCC	
1Ø25	LBCS/LBLO	
1Ø26	LBNE	
1Ø27	LBEQ	
1Ø28	LBVC	
1Ø29	LBVS	
1Ø2A	LBPL	
1Ø2B	LBMI	
1Ø2C	LBGE	
1Ø2D	LBLT	RELATIF

Code hexa	Mnemonique	Mode
1Ø2E	LBGT	RELATIF
1Ø2F	LBLE	RELATIF
1Ø3F	SWI2	INHERENT
1ØB3	CMPD	IMMEDIAT
1ØBC	CMPY	↕
1ØBE	LDY	IMMEDIAT
1Ø93	CMPD	DIRECT
1Ø9C	CMPY	↕
1Ø9E	LDY	
1Ø9F	STY	DIRECT
1ØA3	CMPD	INDEXE
1ØAC	CMPY	↕
1ØAE	LDY	
1ØAF	STY	INDEXE
1ØB3	CMPD	ETENDU
1ØBC	CMPY	↕
1ØBE	LDY	
1ØBF	STY	ETENDU

Code hexa	Mnemonique	Mode
1ØCE	LDS	IMMEDIAT
1ØDE	LDS	DIRECT
1ØDF	STS	DIRECT
1ØEE	LDS	INDEXE
1ØEF	STS	INDEXE
1ØFE	LDS	ETENDU
1ØFF	STS	ETENDU
113F	SWI3	INHERENT
1183	CMPU	IMMEDIAT
118C	CMPS	IMMEDIAT
1193	CMPU	DIRECT
119C	CMPS	DIRECT
11A3	CMPU	INDEXE
11AC	CMPS	INDEXE
11B3	CMPU	ETENDU
11BC	CMPS	ETENDU

Imprimerie de la Manutention à Mayenne
Dépôt légal : novembre 1983
N° d'Éditeur : 3996